Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Diseño de un sistema IOT basado en tarjetas micro:bit y nRF52833DK

Autor: Alejandro Martín Fernández

Tutor: Julio Pastor Mendoza

Curso 2020/2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Diseño de un sistema IOT basado en tarjetas micro:bit y nRF52833DK

Autor: Alejandro Martín Fernandez

Tutor: Julio Pastor Mendoza

Tribunal:

Presidente: D. José Manuel Rodríguez Ascariz.
Vocal 1: Dña. Marta Marrón Romera.
Vocal 2: D. Julio Pastor Mendoza.

Calificación: .....

Fecha: .....

Agradecimientos

A mi familia, cuya educación y valores han hecho que me convierta en la persona que soy a día de hoy, apoyándome en los buenos y malos momentos y haciéndome sentir muy afortunado.

A mi pareja Aida, que siempre me ha escuchado y ha estado en todo momento, siendo un pilar esencial en mi vida y recordándome el valor que tiene uno mismo.

A mis amigos, que han conseguido que me olvidé por un momento del ámbito académico y disfrutase del día a día, afrontando de este modo los días con una sonrisa.

Y por último a todos los profesores de la carrera, cuya formación ha hecho posible adquirir los conocimientos y aptitudes necesarias para realizar el proyecto, en especial a Julio, mi tutor, que ha sabido guiarme en la elaboración y planteamiento del proyecto de un modo espectacular.

Gracias a todos.

Resumen

Este proyecto tiene como principal objetivo construir una red en malla Bluetooth basada en dispositivos micro:bit V2 y nRF52833DK que permite controlar y monitorizar un hogar. Gracias a esto y al uso de diferentes elementos HW, se puede actuar sobre elementos básicos de un hogar como la luz, una puerta o bien actuar en función de aspectos como la temperatura o el gas cuyos valores residen en las características de los servicios Bluetooth. Ligado a este funcionamiento, los datos se transmiten a la nube, usando como gateway el PC, por medio de MQTT. Finalmente, los datos recibidos de MQTT se plasman en una página web de un modo visual y atractivo. El resultado es un sistema IoT robusto, de bajo coste y fácil de usar que está al alcance de cualquier usuario.

Palabras clave

- IoT.
- Micro:bit.
- nRF52833.
- Bluetooth.
- MQTT.

Abstract

The main purpose of this project is to build a Bluetooth mesh network based on micro:bit V2 and nRF52833DK devices which allows to control and monitor a house. Thanks to this and the use of different HW elements, it's possible to act on basic HW elements such as the light, a door or act according to aspects such as the temperature or the gas whose values are stored in the characteristics of the Bluetooth services. Linked to this operation, the data are transmitted to the cloud, using the PC as Gateway, through MQTT. Finally, the received data from MQTT are represented in a web page in a visual and attractive way. The result is a strong, low cost and easy-to-use IoT system which is able to use any user.

Key words

- IoT.
- nRF52833.
- Bluetooth.
- MQTT.
- Models.

Índice

Lista de figuras

Glosario

ADC	Analog-Digital Converter
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
DAC	Digital-Analog Converter
EDR	Enhanced Data Rate
FDMA	Frequency Division Multiple Access
GAP	Generic Access Profile
GATT	Generic Attribute Profie
HCI	Host-Controller Interface
IoT	Internet of Things
I2S	Inter-IC Sound
LMP	Link Management Protocol
L2CAP	Link Control and Adaptation Protocol
MQTT	Message Queuing Telemetry Transport
MTU	Maximum Transfer Unit
nRF5-SDK for mesh	Software Development Kit for mesh from nordic for Series 5
nRF5-SDK	Software Development Kit from nordic for Series 5
PDU	Protocol Data Unit
PWM	Pulse Width Modulation
QoS	Quality of Service
RAM	Random Access Memory
RTC	Real Time Counter
RX	Receptor
SES	Segger Embedded Studio
SDP	Service Discovery Protocol
SM	Security Manager
SPI	Serial Peripheral Interface
TDMA	Time Division Multiple Access
TX	Transmisor
UART	Universal Asynchronous Receiver-Transmitter
UUID	Universally Unique IDentifier
WPAN	Personal Area NetWork

Capítulo 1

Introducción

1.1 Presentación

Con el paso de los años, la necesidad de mantener conectados a la red los objetos cotidianos, ha dado lugar a la aparición de nuevas tecnologías que se engloban dentro del Internet de las cosas, comúnmente llamado IoT. Esto es una realidad que cada vez está más presente en nuestro día a día. No solo es utilizado por las grandes empresas sino también cada vez más por personas que, sin ningún conocimiento previo, hacen uso de sus prestaciones para la monitorización del hogar.

El principal motivo de este auge son las facilidades que ofrecen en la vida cotidiana así como su bajo coste y sencillez, permitiendo al usuario ganar tiempo en el día a día y despreocuparse de cosas tan sencillas como regular la temperatura de su casa.

Para poner en contexto lo mencionado, en el año 2018 se realizó un estudio de la evolución de dispositivos IoT [1] años atrás y se hizó una predicción en los años siguientes. Como se puede observar en la figura 1.1, el número de dispositivos IoT en miles de millones era de 3.8 frente a los 10.1 no IoT en el año 2015. Mientras que los dispositivos convencionales crecían lentamente, la nueva generación creció exponencialmente haciendo la previsión de que para el año 2025 doblarían a los dispositivos no IoT.

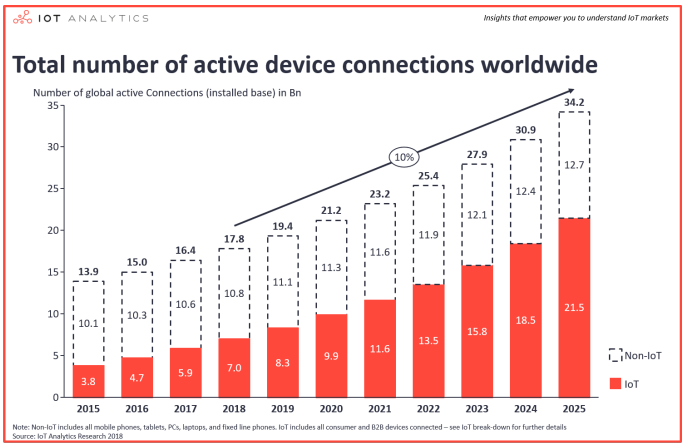

Figura 1.1: Evolución del número de dispositivos IoT. Fuente IOT Analytics Research 2018.

1.2 Motivación

Este proyecto viene motivado por la necesidad que tiene el ser humano de tener controladas situaciones y de una manera más efectiva y rápida. Podemos ver como a lo largo de la historia ha sido así desde la invención de la rueda, que reducía notablemente el esfuerzo, hasta el surgimiento de internet que empezó a sustituir a las enciclopedias. A día de hoy, la necesidad es la monitorización constante de datos ya sean económicos, sociológicos, medioambientales etc.

Por estos motivos y con el fin de lograr un avance se desea desarrollar un sistema casero que sea capaz de conseguirlo, demostrando que no necesariamente se tiene que depender de las grandes empresas sino que con esfuerzo y empeño se consigue.

Una última importante motivación es la de ser capaz de desarrollar un proyecto de IoT, partiendo desde la base más simple hasta un nivel complejo. Con ello, este proyecto lo considero como una de las posibles vías en la que puedo enfocar mi carrera profesional en un sector tan demandado, y aun más gracias a la aparación del 5G. De esta forma, estoy seguro que este proyecto supondrá la adquisición de nuevos conocimientos no solo a nivel profesional de cara al futuro sino también a nivel de satisfacción personal.

1.3 Objetivos

El principal objetivo de la realización de este proyecto es desarrollar un sistema IoT que permite conectar dispositivos basados en tarjetas micro:bit y nRF52833DK, usando como Gateway un ordenador de tal modo que se consiga construir una red en bluetooth en malla que permite además realizar una monitorización de datos. La comunicación de datos se realizará vía Bluetooh de bajo consumo (BLE) , mientras que la operatividad en la nube se realizará por medio del protocolo MQTT que permitirá transmitir los datos a una página web.

Su aplicación es la monitorización de datos de una forma cómoda, visual y atractiva en un entorno determinado como puede ser una casa. Permite acceder a los datos representados en visualizaciones por medio de una página web de una forma sencilla, pudiendo ser usado el sistema por cualquier usuario independientemente de los conocimientos que tenga.

De esta manera, con este proyecto no solo se pretende entender como construir un sistema IoT, sino también entender a fondo el protocolo de comunicación Bluetooth y presentar los datos de una forma atractiva, consiguiendo que los alumnos más jovenes se introduzcan en esta tecnología de una manera amena, divertida y visual.

Otro objetivo importante es demostrar que es posible la construcción de un sistema estable, robusto y barato por medio de dispositivos accesibles por todo el mundo y haciendo uso de plataformas de uso libre y sin coste.

Un último objetivo realmente importante es el referente al uso de los microcontroladores. En este caso, se ha optado por usar micro:bit y nRF52833DK lo que ha implicado un profundo aprendizaje para su utilización. Como posteriormente se detallará, este tipo de microcontroladores se utilizan sobre todo en el ámbito de la docencia (sobre todo las micro:bit para los mas jóvenes).

1.4 Estructuracion del proyecto

En este apartado se va a presentar el flujo de trabajo que se ha seguido para la realización exitosa del proyecto. Gracias a esta estructuración se logra una mejor organización y productividad además de asentar los conocimientos en diferentes fases y de manera organizada. Las etapas que comprenden este esquema son las siguientes:

- **Etapa 1.** Durante la fase inicial es importante hacer un planteamiento inicial de como se hará el proyecto, qué elementos Hardware se requieren, qué tecnologías se utilizan, qué herramientas Software se van a usar. Esta fase es esencial en la realización de todo proyecto ya que permite ahorrar tiempo de cara al futuro y mentalizarse de como lograr lo propuesto.
- **Etapa 2.** Una vez hecho el planteamiento inicial de todo lo necesario, se investiga el funcionamiento de los diferentes microcontroladores, y los entornos que se van a utilizar. Con diferencia esta será la fase más compleja y larga ya que no se disponen de conocimientos previos de como utilizar los diferentes entornos ni elementos.
- **Etapa 3.** Con lo anterior realizado, se procede a aplicar las herramientas Software con ejemplos sencillos y de manera paralela se investiga el protocolo de comunicación Bluetooth, esencial y clave en el cumplimiento del proyecto.
- **Etapa 4.** Con los conocimientos adquiridos de BLE se empiezan a desarrollar ejemplos básicos y servicios propios de BLE. Además se aprende a como monitorizar y controlar los datos para poder plasmarlos de manera gráfica en el entorno que se vaya a usar.
- **Etapa 5.** Teniendo los conocimientos propios de Bluetooth se procede a escalar a un nivel más en este protocolo de comunicación, investigando *BLE for mesh* para el diseño del propio sistema en el que consistirá el proyecto.
- **Etapa 6.** Llegados a este punto y lo difícil ya realizado, se plasman los datos del sistema en una interfaz visual, usando el PC como gateway gracias a MQTT.
- **Etapa 7.** Finalmente, se realiza una documentación especificando los pasos realizados para el cumplimiento del proyecto con éxito así como evidencias del resultado. Esta etapa a pesar de qué es la última se realizará de manera paralela a partir de la Etapa 4 en la que ya se han adquirido todos los conocimientos necesarios para el desarrollo correcto. Sin embargo, hasta que no se termine la Etapa 6 no se podrá completar la memoria en su totalidad.

1.5 Estructuración de la memoria

En este último apartado se tiene como intención explicar como se ha decidido organizar la memoria con el fin de lograr una comprensión completa del proyecto tanto en aspectos teóricos como prácticos del tal modo que se logre alcanzar cada uno de los objetivos mencionados. De esta manera, la memoria del proyecto puede dividirse de la siguiente manera:

- **Bloque de teoría**
  En él, se abordará el estándar de comunicación Bluetooth y su pila de protocolos. Además se entrará en profundidad con el funcionamiento de *Bluetooth Low Energy* y *Bluetooth mesh* que formará la base del proyecto. También se explicará el protocolo de comunicación MQTT, esencial para poder realizar la monitorización de datos. Posteriormente, una vez detallado estos protocolos y su funcionamiento se pasa a un aspecto más concreto en el que se explica los elementos HW utilizados y sus características. Finalmente se explica el SW utilizado así como una preparación del mismo para su correcto uso.

- **Bloque práctico**
  En este bloque se explicará el desarrollo del sistema propuesto, su código, detalles a tener en cuenta y evidencias del funcionamiento. Finalmente se concluirá con un debate de los resultados.

- **Conclusión**
  Por último la memoria queda finalizada con una conclusión acerca del proyecto y vistas a futuro del mismo. Además, se ofrece un manual de usuario para el correcto uso e instalación del sistema en el que cualquier lector podrá tener acceso al código fuente así como a una serie de vídeos que enseñan a utilizarlo.

Capítulo 2

Bluetooth

Como se ha mencionado en el capítulo 1, uno de los objetivos de este proyecto es entender el funcionamiento de uno de los estándares de comunicación inalámbrica más conocidos, *Bluetooth*, y dado que es la base del sistema que se va a desarrollar, su profundo conocimiento es más que obligatorio. En este capítulo se tiene como propósito explicar en qué consiste y cada una de las capas que lo conforman (*Bluetooth Classic*). Posteriormente se explicará *Bluetooth Low Energy* y finalmente se explicará una extensión del mismo, *Bluetooth Low Energy for mesh*, creado sobre la pila de protocolos de *Bluetooth Low Energy* y que permite el diseño de redes en malla en base a unos tipos de modelos, siendo el usado en el sistema.

Figura 2.1: *Bluetooth.* Fuente *Bluetooth SIG*

2.1 ¿Qué es Bluetooth?

Bluetooth [2][3] se trata de una especificación para redes inalámbricas de área personal (WPAN) creado por *Bluetooth Special Interest Group Inc* en los años 90, pero no fue hasta el año 2002 cuando se ratificó como estándar *IEEE 802.15.1-2002* en donde se solucionaron una gran cantidad de errores de su versión inicial. Consiste en un estándar inalámbrico que permite la transmisión de datos entre dispositivos a corto alcance, facilitando las comunicaciones entre ambos, eliminando la presencia de cables o conectores y permitiendo una interacción sencilla y rápida entre los dispositivos por medio de un radioenlace en la banda ISM de los 2.4GHz.

2.1.1 Breve resumen de su historia desde sus inicios hasta hoy

Bluetooth nace durante los años noventa gracias a dos trabajadores suecos de Ericsson, Jaap Haartsen y Mattisson Sven, que buscaban una tecnología que permitiese la comunicación a corto alcance entre dispositivos y que no conllevase una gran energía como sustituto del cable. Aunque *Ericsson* empezó a investigar sobre esta tecnología a inicios de los años 90, no fue hasta el año 1998 cuando un consorcio formado por las compañías Ericsson, Nokia, Toshiba e Intel formaron el Bluetooth Special Interest Group o Bluetooth SiG. Posteriormente en 2002 fue reconocido como estándar, ganando la atención de grandes empresas. En un principio, Bluetooth era capaz de transmitir datos a una velocidad de 720 kbs pero la velocidad ha mejorado notablemente y también el alcance y la distancia a la que deben estar los dispositivos entre sí. De este modo sus puntos más notables en su largo recorrido han sido los siguientes:

- Bluetooth v1.1 (2002): Se reconoce como estándar y se resuelven numerosos problemas de la versión 1.0.
- Bluetooth v1.2 (2003): Es compatible con USB 1.1 (Universal Serial Bus) y se consigue una mayor velocidad de transmisión además de añadirse la Host Controller Interface (HCI).
- Bluetooth v2.0 + EDR (2004): Cuenta con una tasa EDR de 3 Mbit/s y una tasa de transferencia de datos práctica de 2,1 Mbit/s.
- Bluetooth v3.0 + HS xxx (2009): Se añade 802.11 como transporte de alta velocidad y permite el uso de alternativas MAC y PHY para el transporte de datos de perfil Bluetooth.
- Bluetooth v4.0 (2010): Se incluye una implementación nueva, *Bluetooth Low Energy*, dentro del estándar como un subconjunto de Bluetooth v4.0 con una pila de protocolo completamente nueva para desarrollar rápidamente enlaces sencillos gracias a CSR, Nordic Semiconductor y Texas Instruments que impulsaron esta idea. Esta idea dará lugar a la distinción entre *Bluetooth Classic* y *Bluetooth Low Energy*.
- Bluetooth v5.0 (2016-2017): Doble de velocidad, mejor fiabilidad y rango de cobertura; además de contar con 8 veces más de capacidad.
- Bluetooth v5.1 (2019): La principal novedad que presenta está en que se puede saber la ubicación de otros dispositivos a los que estén conectados.
- Bluetooth v5.2 (2020): Mejoras importantes en el modo de radiofrecuencia Bluetooth LE (Low Energy) y mayor seguridad.

2.1.2 Pila de protocolos

Definido como un estándar, Bluetooth se compone de una arquitectura sustentada en una serie de capas que permite a todo dispositivo realizar un establecimiento y control de enlace entre dispositivos y controlar el acceso de datos para posteriormente encapsularlos y transmitirlos como paquetes IP sobre la pila de protocolos TCP/IP hasta llegar a la capa de aplicación. Todo protocolo Bluetooth debe incluir al menos las siguientes capas LMP, L2CAP y SDP. La pila de protocolos es la presentada en la figura 2.2:

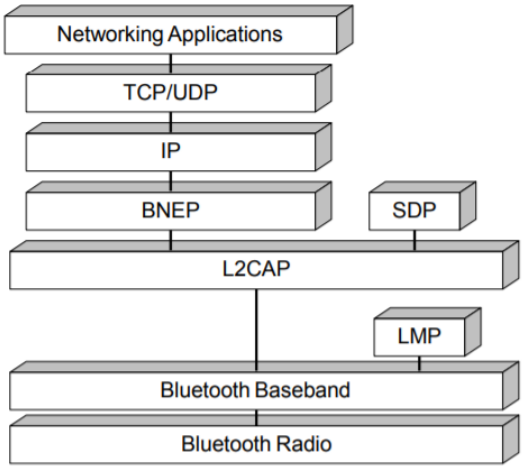

Figura 2.2: Pila protocolos *Bluetooth*. Fuente *Bluetooth SIG, "BLE core specification"*

LMP

Link Management Protocol es un protocolo de control de enlace que se usa para el establecimiento y control del enlace de radio entre dos dispositivos. Está implementado en el controlador y trabaja sobre la frecuencia ISM de 2.4GHz.

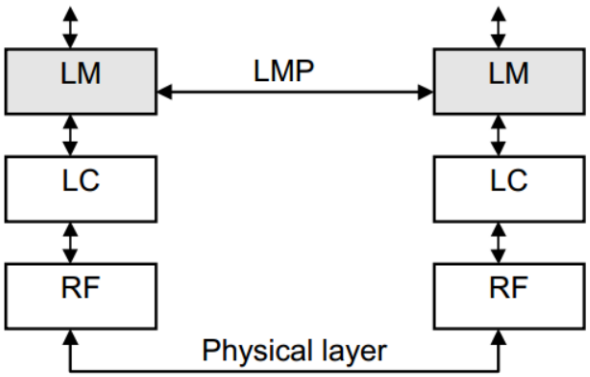

Figura 2.3: Capa LMP. Fuente *Bluetooth SIG, "BLE core specification"*

L2CAP

Logical Link Control and Adaptation Protocol(L2CAP) es la capa de control de acceso de datos Bluetooth. Es usada para multiplexar múltiples conexiones lógicas entre dos dispositivos, proporcionando segmentación y reensamblado de paquetes. Además proporciona a los paquetes una carga útil que se puede configurar hasta 64 Kbps, y con una MTU por defecto de 672 bytes. En la figura 2.4 se ilustra los eventos y acciones realizados por la capa L2CAP, en donde el cliente inicia una solicitud y el servidor responde.

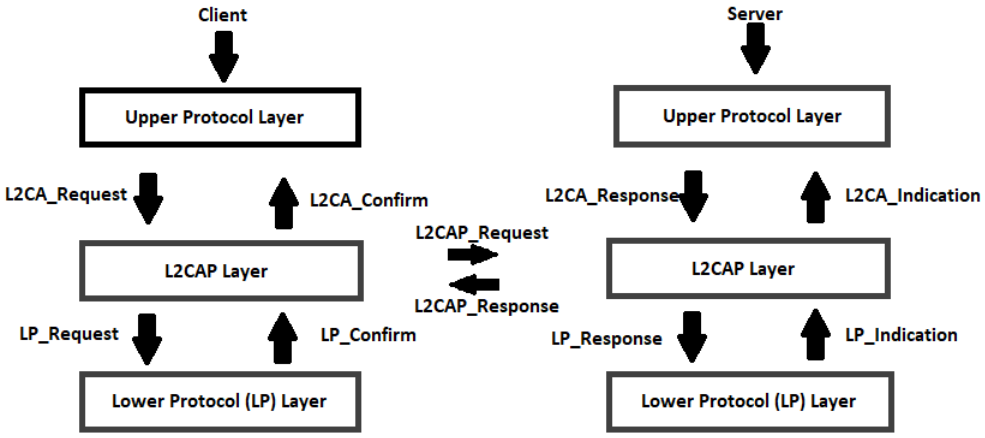

Figura 2.4: Capa L2CAP. Fuente *Bluetooth SIG, "Bluetooth core specification"*

SDP

Service Discovery Protocol(SDP) permite a un dispositivo descubrir los parámetros y servicios asociados de otros dispositivos, determinando su perfil de Bluetooth. Cada servicio de un dispositivo viene identificado con un UUID. Un servicio ya creado viene registrado en una base de datos, sin embargo también se pueden implementar como se verá durante el desarrollo del sistema en el capítulo 7.

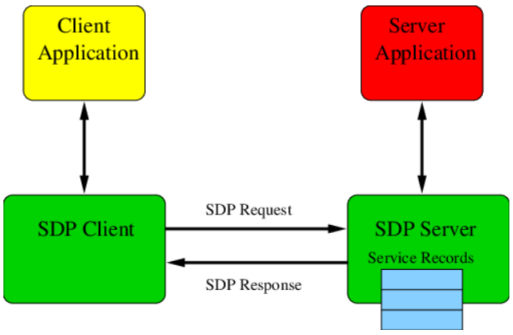

Figura 2.5: Capa SDP. Fuente *Bluetooth SIG*

BNEP

Bluetooth Network Encapsulation Protocol(BNEP)[4] es el protocolo de encapsulamiento del estándar Bluetooth. Su principal propósito es la transmisión de paquetes IP de control en un perfil de red de área personal, ofreciendo características similares a Ethernet.

Como nota, es necesario comentar que la pila de protocolos Bluetooth adopta el uso de la pila de protocolos TCP/IP explicado a lo largo de la carrera. Sin embargo, su explicación se sale fuera del objetivo de este proyecto. A continuación se muestra una imagen de toda cabecera BNEP:

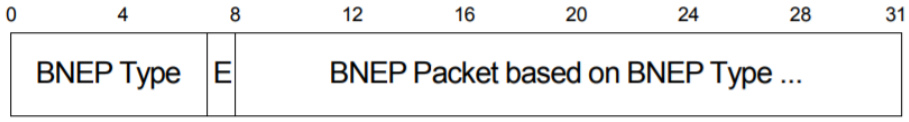

Figura 2.6: Cabecera BNEP. Fuente *Bluetooth SIG, "Bluetooth core specification"*

Haciendo referencia a la figura 2.6, los posibles valores que puede tomar *BNEP Type* se muestran en la siguiente figura:

Value	BNEP Packet Type
0x00	BNEP_GENERAL_ETHERNET
0x01	BNEP_CONTROL
0x02	BNEP_COMPRESSED_ETHERNET
0x03	BNEP_COMPRESSED_ETHERNET_SOURCE_ONLY
0x04	BNEP_COMPRESSED_ETHERNET_DEST_ONLY
0x05 – 0x7E	Reserved for future use
0x7F	Reserved for 802.2 LLC Packets for IEEE 802.15.1 WG

Figura 2.7: Tipos BNEP. Fuente *Bluetooth SIG, "Bluetooth core specification"*

Características principales

Explicado el estándar Bluetooth y sobre qué se compone y sustenta es necesario mencionar las características de la versión actual (v5.2) con el fin de reflejar su potencial y defender su actual utilización frente otros estándares de comunicación como *Zigbee*. Las características principales de su última versión son:

- Rango frecuencia ISM 2.4 GHz.
- Combinación de dos métodos de acceso múltiple para compartir el medio de radio, FDMA y TDMA.
- Rango distancia de hasta 240 metros.
- Velocidad de transferencia de hasta 50Mbps.
- Comprensión audio optimizada con soporte de más 2 dispositivos de manera simultánea *(LE Isochronous Channels)*
- Comunicación cifrada.
- Mayor estabilidad de la señal al permitir optimizar dinámicamente la potencia de la transmisión (LE Power Control) y notable disminución del consumo.

2.2 *Bluetooth Low Energy*

En puntos anteriores se ha tratado qué es Bluetooth, en qué se basa y sus principales características. En este, se pretende explicar el concepto *Bluetooth Low Energy (BLE)* y sobre cuya base se construirá el sistema. Como se mencionó anteriormente en *Breve resumen de su historia desde sus inicios hasta hoy*, BLE salió a la luz como una implementación nueva de Bluetooth v4.0. Su objetivo era poner fin al problema que enfrentaba la transferencia continua de datos vía Bluetooth ya que en muchos casos se requería una notable fuente de energía. Con esta idea presentada, los principales fabricantes de semiconductores, vendedores de dispositivos y proveedores de servicio como *Nordic Semiconductor*, cuyos procesadores son los que se van a usar, lograron impulsar esta extensión, estando a día de hoy implementada en la mayoría de dispositivos móviles. Dicho esto, a partir de entonces se conocieron dos conceptos, *Bluetooth Classic* y *Bluetooth LE*.

2.2.1 Diferencias entre *Bluetooth Classic* y *Bluetooth Low Energy*

A pesar de ser una implementación adicional, BLE presenta importantes diferencias respecto a su predecesor[5]. El primer punto radica en que requiere menos energía. Esto es debido a que mientras la versión clásica está permanentemente activa, BLE permanece en modo de suspensión constantemente, excepto cuando se inicia una conexión la cual además se realiza de manera más rápida. Esto tiene notables consecuencias ya que la batería puede alcanzar varios años de vida. Otra notable diferencia son los propósitos en que se usan estas tecnologías, mientras que Bluetooth se usa para manejar muchos datos (como transferencia de archivos), BLE se usa para aplicaciones en donde la transferencia es menor. A esto se le añade la posibilidad adicional que BLE presenta, permite la creación de diferentes topologías de red entre las que destaca la topología en malla, usada en este proyecto.

Figura 2.8: Topologías. Fuente *Bluetooth SIG*

Los canales de frecuencias ya no son los mismos. BLE usa 40 canales espaciados 2 MHz de los cuales 3 son *advertising* y 37 son canales de datos, frente a los 79 canales de la versión clásica espaciados 1 MHz.

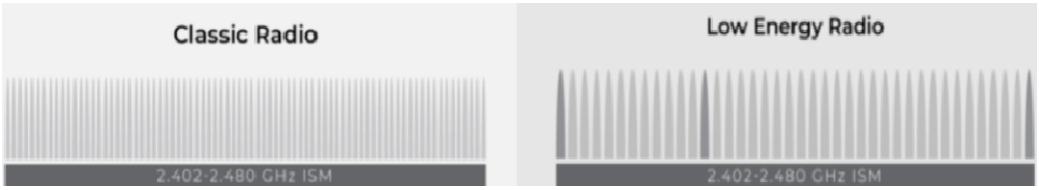

Figura 2.9: Distribución de canales. Fuente *Bluetooth SIG*

Por ultimo, una diferencia reseñable reside en qué la pila de protocolos BLE es completamente nueva con respecto a su predecesora debido al cambio de paradigma. A continuación, se ofrece la pila de protocolos extraída de los procesadores *Nordic* y que será la que se usará como base para la elaboración del proyecto[6]:

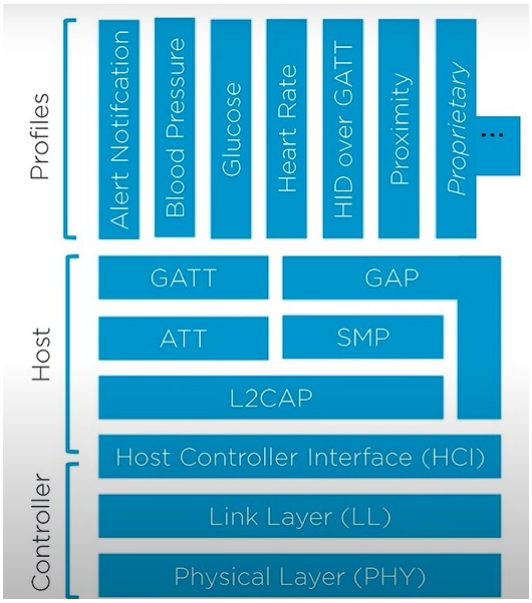

Figura 2.10: *BLE stack*. Fuente *Nordic Semiconductor*

En ella se distinguen tres partes diferenciadas. Por un lado, la parte del controlador que engloba la capa física, la de enlace y la interfaz HCI. Por otro lado, la parte del Host en la qué se definen las características, el perfil Bluetooth del dispositivo, la capa de control de acceso y la de seguridad. Finalmente, toda la parte relativa del Host conecta con la aplicación en la que se definen los servicios haciendo uso de lo mencionado.

2.2.2 Pila de protocolos BLE

En este punto se van a explicar las diferentes capas de BLE presentada en la figura 2.10[7] que permitirá no solo entender el estándar sino también servir de base para la comprensión de un nivel más, *BLE for mesh*, y en definitiva entender el desarrollo del código que se va a utilizar para el cumplimiento exitoso del proyecto. Para información adicional de la especificación, descargar la especificación completa en [6] como se muestra en la siguiente figura:

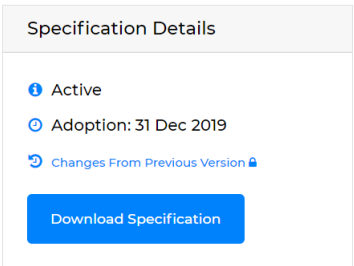

Figura 2.11: BLE core specification. Fuente *Bluetooth SIG*

Capa física

Trabaja con una modulación GFSK en la banda de los 2.4GHz hasta 2.4835GHz dividida en 40 canales de radiofrecuencia de los cuales 3 son utilizados para el broadcasting y 37 con propósitos para la comunicación (dispone de velocidades de transmisión de 1 o 2 Mbps).
F = 2402 +k*2MHz; K = 0...39)

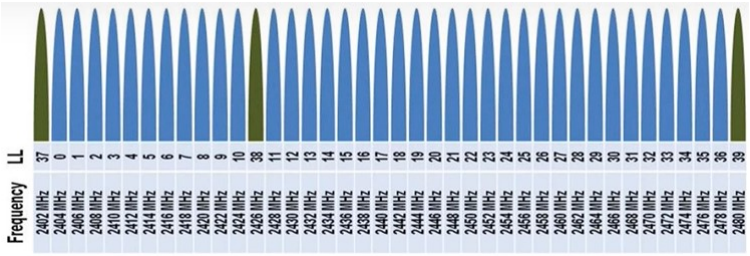

Figura 2.12: Canales BLE. Fuente *Bluetooth SIG*

Capa de enlace

La capa de enlace es la responsable de anunciar datos, escanear dispositivos, establecer y mantener las conexiones además de encargarse de que los paquetes se transmitan en orden. Su funcionamiento se agrupa en la máquina de estados ofrecida en la figura 2.13.

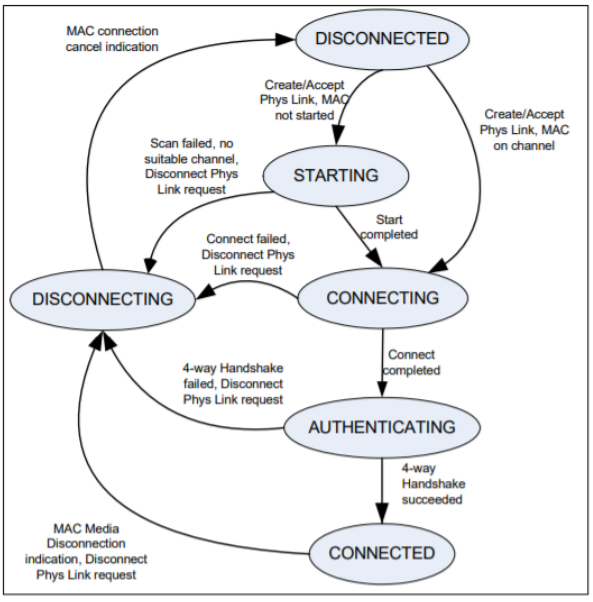

Figura 2.13: Maquina estados capa enlace. Fuente *Bluetooth SIG, "Bluetooth core specification"*

- **DISCONNECTED**: Capa de enlace no activa. Es el estado inicial.
- **STARTING**: Se ha seleccionado un canal y el control de acceso se está inicializando.
- **CONNECTING**: El dispositivo inicializado espera mensajes de otro dispositivo.

- **AUTHENTICATING**: Los dispositivos realizan el proceso de seguridad asociado.
- **CONNECTED**: Un enlace seguro se ha establecido con el dispositivo remoto.
- **DISCONNECTING**: El control de acceso completa la desconexión y se regresa al estado inicial.

HCI

Interfaz encargada de comunicar el bloque *Host* con el bloque *Controller* gracias al uso de APIs o HW interfaces como la UART, USB que controlan esta interfaz por medio de comandos. El bloque controlador envía datos y eventos a través de HCI al host. El host envía comandos y datos al controlador. La siguiente figura muestra un ejemplo de cómo la interfaz *UART* realiza esta tarea.

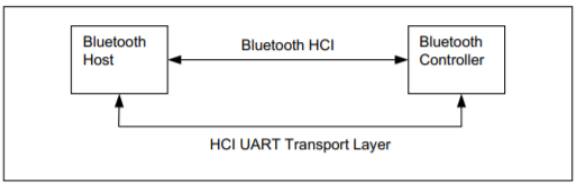

Figura 2.14: HCI-UART. Fuente *Bluetooth SIG, "BLE core specification"*

En ella se distinguen, un total de 5 tipos de paquetes que pueden ser enviados por medio de la capa de transporte UART:

HCI packet type	HCI packet indicator
HCI Command packet	0x01
HCI ACL Data packet	0x02
HCI Synchronous Data packet	0x03
HCI Event packet	0x04
HCI ISO Data packet	0x05

Figura 2.15: Tipos de paquetes. Fuente *Bluetooth SIG, "BLE core specification"*

Las características de esta interfaz por medio de UART se ofrecen a continuación en donde:

- Si **CTS = 1**, HCI permite enviar paquetes.
- Si **CTS = 0**, entonces HCI no envía paquetes.

Baud rate:	manufacturer-specific
Number of data bits:	8
Parity bit:	no parity
Stop bit:	1 stop bit
Flow control:	RTS/CTS
Flow-off response time:	manufacturer specific

Figura 2.16: HCI-UART. Fuente *Bluetooth SIG, "BLE core specification"*

L2CAP

Capa encargada de encapsular los datos de los servicios y permitir una comunicación de datos e2e. Dado que ya se ha explicado anteriormente, no se entrará en mayor detalle.

SMP

Security Manager Profile es la capa encargada de realizar una distribución de claves que realiza las funcionalidades de identificación y encriptación por medio de un cifrado AES-128. La generación de las claves no afecta a otros dispositivos.

ATT

Attribute protocol, permite a los dispositivos leer o escribir en una característica contenida en un servidor. Cada valor almacenado se define como un **atributo**, identificados por un identificador único llamado **UUID** cuyo valor máximo es de 128 bits. Los mensajes de ATT son enviados sobre la capa L2CAP.
ATT define dos roles, Cliente y Servidor, siendo posible que un dispositivo sea un *ATT Server y ATT Client*. El *ATT Server* almacena los atributos y acepta solicitudes ATT, comandos y confirmaciones de un *ATT Client*. También un *ATT Server* es capaz de enviar indicaciones y notificaciones de los datos de manera asíncrona al cliente cuando un evento ocurra, evitando que este realice solicitudes continuamente.

GATT

Generic Attribute Profile, está construido sobre la parte más alta de *Attribute protocol* y se encarga de establecer y realizar operaciones comunes así como definir un *framework* para el transporte de datos almacenados por *Attribute protocol*. GATT define dos roles, Cliente y Servidor, y da un formato a los datos de ATT como servicios y características. Un **Servicio** puede contener una colección de características, mientras que una **Característica** contiene un único valor y una serie de propiedades que hacen que el valor sea de lectura, escritura, notificación o indicación.

GAP

Generic Access Profile, es el responsable de manejar la funcionalidad base de todos los dispositivos Bluetooth, como por ejemplo los métodos de acceso de los diferentes dispositivos así como establecer el inicio y fin de conexión. También se encarga de tareas como el descubrimiento de dispositivos y servicios de otros dispositivos. Por último una tarea muy importante que realiza es definir 4 roles específicos, permitiendo a los dispositivos optimizarse según sea su rol:

- ***Broadcaster***: Optimizado para aplicaciones solo de transmisión de datos. No establece conexiones.
- ***Obserser***: Optimizado para aplicaciones solo de recepción de datos. No establece conexiones.
  Un ejemplo de como actúan ambos roles se muestra a continuación:

Figura 2.17: *Broadcaster - Observer*. Fuente

- ***Peripheral***: Optimizado para dispositivos que soportan una única conexión. Tienen un rol de esclavos con respecto al rol *Central*.
- ***Central***: Soporta múltiples conexiones y es el inicializador para todas las conexiones con los periféricos. Tienen un rol de master con respecto al rol *Peripheral*.
  Un ejemplo de como actúan ambos roles se muestra a continuación:

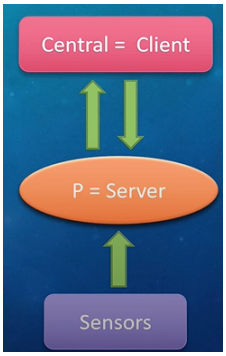

Figura 2.18: *Central - Peripheral*

De forma resumida, se puede observar que todo dispositivo BLE cuenta con al menos dos perfiles, uno encargado de definir los parámetros de conexión así como el descubrimiento de los servicios (GAP) y otro enfocado en definir los servicios que provee.

2.2.3 Servicios y características BLE

Como se ha explicado, todo perfil BLE se compone de uno o más servicios (cada uno de ellos con su propio UUID) que dan lugar a la funcionalidad básica de un dispositivo. Dentro de cada servicio se definen una o varias características BLE. En ellas se almacenan valores de variables que pueden ser sensores, estados de un elemento HW entre otros. A su vez, toda característica BLE viene formada por 3 entidades; el valor, el UUID (Identificador Único Universal) y una serie de propiedades entre las que destacan escritura, lectura y notificación. Un ejemplo de un Servicio BLE en un dispositivo puede ser por ejemplo el cálculo de glucosa de una persona. Este servicio BLE puede representar valores como son el ritmo cardíaco y el nivel de glucosa cuyos valores se guardan en características diferentes. Ambas vienen definidas por un UUID, por ejemplo 0x1520 y 0x1521 respectivamente, el valor y la propiedad de lectura y notificación.

Con el objetivo de facilitar una mejor comprensión de lo explicado, se ofrece una imagen representativa en donde se puede ver como un perfil, definido en un dispositivo, contiene uno o más servicios. Estos servicios (identificados por un UUID), engloban un conjunto de características (identificadas también por un UUID) compuestas por unas propiedades (definidas por ATT protocol, pueden ser escritura, lectura, indicacion y notificacion), un valor y uno o varios descriptores.

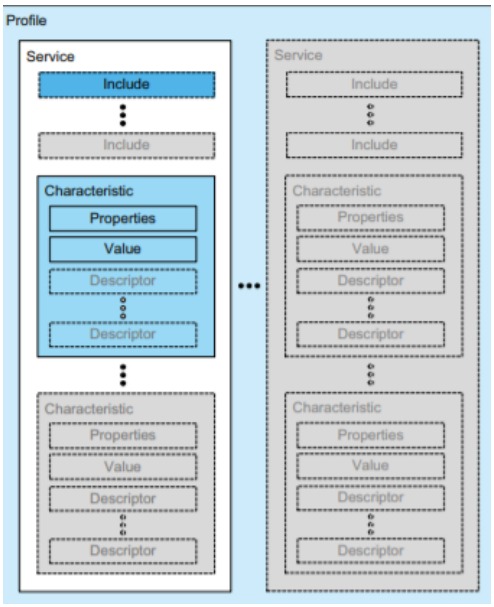

Figura 2.19: Perfiles-Servicios-Características. Fuente *Bluetooth SIG, "BLE core specification"*

2.3 *Bluetooth Low Energy for mesh*

Como se ha visto, Bluetooth ha estado en constante desarrollo desde sus inicios dominando la tecnología inalámbrica para productos relacionados al audio y periféricos. En el año 2010, Bluetooth dió un paso de gigantes con la introducción de las redes inteligentes, empezándose a centrar no unicamente en dispositivos de audio sino también relacionados con el hogar, la salud y el estilo de vida. Con ello surgió *Bluetooth Low Energy* que evolucionó poco a poco, surgiendo hasta lo que día de hoy conocemos redes Bluetooth, entre las que destaca las redes en malla. Basado en BLE y construido sobre su parte más alta, *BLE for mesh*[8] define un estándar nuevo que, por medio de una serie de elementos. permite construir una red en malla. De este modo, mientras que *BLE* ofrece una comunicación *one to one*, *BLE for mesh* provee una comunicación *many to many*. Con ello surge un nuevo estándar que además coexiste con el estándar BLE, ofreciendo un grandísimo potencial.

En este apartado se explica en qué consiste una red en malla, el conjunto de nodos que la constituyen, su pila de protocolos *BLE mesh* así como un proceso esencial en toda red *mesh*, el aprovisionamiento, que hace posible su funcionamiento, dando lugar a esta tecnología.

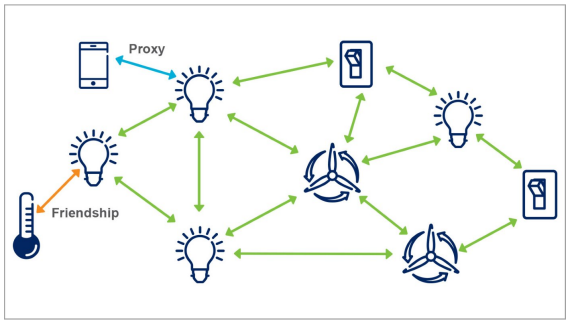

Figura 2.20: Ejemplo *BLE mesh*. Fuente *digikey, "designing bluetooth low energy smart applications part 1"*

2.3.1 Conceptos básicos

Dispositivos y nodos

Dentro de una red *mesh*, distinguimos entre dispositivos y nodos. Un nodo es todo aquel dispositivo que ha sido aprovisionado, es decir que ha pasado un proceso para poder ser utilizado en la red. Todo aquel que no ha sido aprovisionado o lo ha dejado de ser no es un nodo sino un dispositivo.

Elementos

Dentro de la red, cada nodo puede tener diferentes partes que son controladas de manera independiente. Por ejemplo, en un microcontrolador (nodo) podemos tener dos elementos, un servomotor y un led.

Estados

Los elementos dentro de una red *mesh* pueden encontrarse en diferentes estados cuyo valor desemboca en el desencadenamiento de una acción u otra. Por ejemplo, en un led (elemento) se pueden distinguir dos estados, ON y OFF.
Estos estados, a su vez, vienen caracterizados por propiedades, indicando si son de solo lectura, solo de escritura o ambos.

En el paso de un estado a otro, se distingue un tipo de estado adicional llamado **estado de transición**.

Mensajes

Los mensajes son el mecanismo mediante el cual se invocan las operaciones en la malla y por tanto interactúan con los **estados** de un **elemento**, representando un estado o una colección de ellos. Se distinguen 3 tipos de mensajes:

- ***SET***: Se solicita la escritura de un estado. Por ejemplo enviar un 1 a un pin del microcontrolador.
- ***GET***: Se solicita la lectura de un estado. Por ejemplo, comprobar el porcentaje de luminosidad de un LDR.
- ***STATUS***: Mensajes que se envían en respuesta a mensajes *GET* o *SET*.

Direcciones

Como se ha comentado, los mensajes desencadenan que un elemento cambie de estado. Para ello, es necesario que estén encaminados a ciertas direcciones, siendo estas una representación lógica de un nodo. Se distinguen varios tipos de direcciones entre las que destacan:

- **Direcciones Unicast**: Hacen referencia a un único nodo.
- **Direcciones de grupo**: Hacen referencia a un conjunto de nodos.

Se pueden realizar dos acciones con respecto a los mensajes, publicar o suscribirse. De este modo, un nodo puede publicar mensajes a uno/varios nodos o bien atender una solicitud.

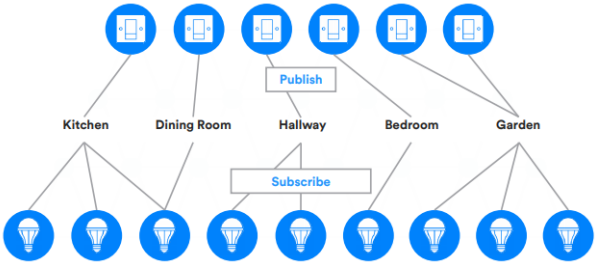

Figura 2.21: Mapeo de direcciones. Fuente *Bluetooth SIG, "Mesh Technology Overview"*

Modelos

Todos los conceptos básicos anteriormente mencionados se relacionan en torno a los modelos. La funcionalidad de un elemento viene dada por uno o varios modelos dentro de una red *mesh*. Se distinguen un total de 3 tipos de modelos:

- **Modelo servidor**: Define una colección de estados, estados de transición y mensajes que el elemento contenido en el modelo pueden enviar o recibir. También define el comportamiento de los mensajes y estados.

- **Modelo cliente**: A diferencia del modelo servidor no define estados, sino qué define mensajes de *SET*, *GET* o *STATUS* de cara a un estado de un modelo servidor.

- **Modelo de control**: Contienen ambos modelos, un modelo servidor para permitir la comunicación con otro cliente y un modelo cliente que permite la comunicación con servidores modelo.

Se ofrece un ejemplo de cómo un elemento puede tener diferentes modelos:

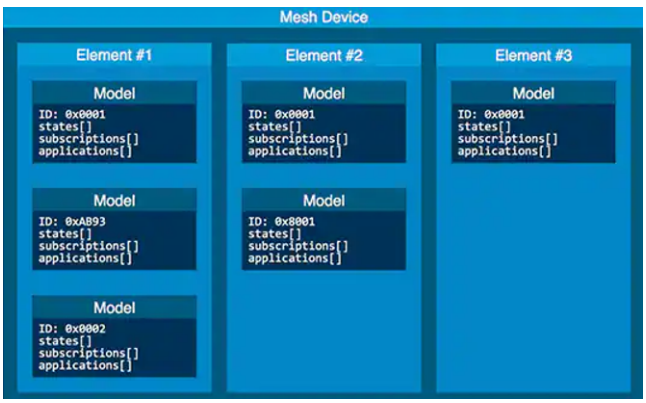

Figura 2.22: Ejemplo relación modelo - elemento. Fuente *digikey, "designing bluetooth low energy smart applications part 1"*

Escenas

Se tratan de un conjunto de estados cuyos valores cambian a la vez con el fin de lograr un funcionamiento simultáneo de diferentes elementos. Por ejemplo, una escena puede ser el modo cine. En ella se apagan las luces, se baja la persiana y se enciende la televisión.

Aprovisionamiento

Como se ha mencionado con anterioridad, para que un dispositivo pasé a ser un nodo es necesario que se lleve a cabo un proceso conocido como **aprovisionamiento**, esencial y necesario para que la red *mesh* funcione.

El aprovisionamiento se trata de un proceso a partir del cual un dispositivo se une a la red *mesh* y se convierte en nodo. Consiste en varias fases que además de permitir constituir la red, aporta una seguridad adicional gracias al intercambio de claves generadas, en donde todos los nodos comparten la misma clave. Las fases que lo componen son:

- **1.*Beaconing***
  En esta primera fase, un dispositivo (no aprovisionado) indica dicho estado, usando datos de publibicidad *mesh* o *Mesh Beacon advertises*.

- **2. Invitación**
  Una vez indica su presencia, el aprovisonador envía la invitación al dispositivo para ser aprovisionado y convertirse en nodo en forma de una PDU quien responde de igual manera con otra PDU.

- **3. Intercambio de claves**
  Aceptada la invitación se realiza un intercambio de claves públicas, usando el método OBB(*out-of-band*).

- **4. Autenticación**
  El dispositivo que se va a aprovisionar genera una salida aleatoria, de uno o varios dígitos, utilizando una acción adecuada a sus capacidades. Por ejemplo, un parpedeo de LEDs. El usuario ingresa los dígitos generados por el nuevo dispositivo en el aprovisionador y se lleva a cabo un intercambio criptográfico entre los dos dispositivos, que implica el número aleatorio, para completar la autenticación de cada uno de los dos dispositivos al otro.

- **5. Distribución de datos aprovisionados** Completada la autenticación se genera una clave de sesión en el dispositivo que se usa para completar el proceso, obteniendo los datos aprovisionados y una clave de red. Finalmente, se descubren los servicios de este dispositivo ya aprovisionado, convertido en nodo, en donde se incluyen modelos y elementos.

  Nota: Esta acción puede ser llevada a cabo por un microcontrolador o un dispositivo Android o iOS por medio de la app *NRF Mesh*

2.3.2 Características de los nodos

Dentro de una red *mesh*, un nodo puede desempeñar una o varias características o papeles que refuerzan el funcionamiento de la red desarrollada. Gracias a estas características un nodo podría, además de cumplir con todas sus funcionalidades, retransmitir a otros nodos aumentando el alcance Bluetooth entre otras cosas. Las posibles características en *BLE mesh* son:

- ***Low Power and Friend feature***: Se tratan de dos características diferentes pero que siempre van unidas. En ocasiones, los dispositivos tienen una fuente de alimentación limitada y es necesario extender su ciclo de vida lo máximo posible. Por este motivo, está la característica *Low Power*. Gracias a ella, un nodo se duerme y despierta de manera periódica cada cierto tiempo, preguntando a un nodo con el que haya establecido amistad, *Friend node*, que ha ocurrido en la red. Con esto, el nodo amigo almacena los mensajes dirigidos al LPN y los entrega cada vez que el LPN sondea al nodo amigo.

  NOTA: La relación entre nodo LPN y nodo amigo es de N:1

- ***Relay feature***: Permite a un nodo retransmitir mensajes dentro de la malla Bluetooth, logrando extender el alcance de la comunicación de la red y haciendo que la comunicacion se realice en diferentes saltos.

- ***Proxy feature***: Capacidad que tiene un nodo para intercambiar mensajes entre *advertising bearer* y *GATT bearer*, puesto que exponen una interfaz GATT que los dispositivos Bluetooth LE pueden utilizar para interactuar con la red en malla.

A continuación se muestra una figura con el proceso de aprovisionamiento y los roles de los nodos:

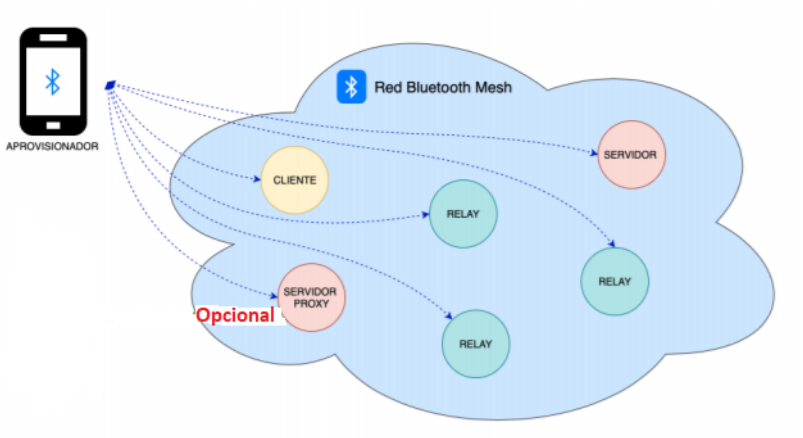

Figura 2.23: Ejemplo red malla BLE con roles.

2.3.3 Pila de protocolos *BLE for mesh*

Como se ha comentado anteriormente, la pila de protocolos *BLE for mesh* se desarrolla en la parte más alta del *stack* BLE, coexistiendo y trabajando de manera conjunta para lograr no solo la interacción entre dispositivos sino también la monitorización de los servicios y características[8].Al igual que para *Bluetooth Classic* y *Bluetooth Low Energy* se va a explicar las diferentes capas que componen el estándar *BLE mesh* mostrados en la figura 2.24:

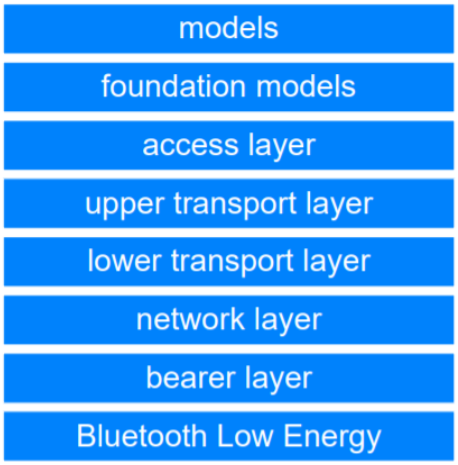

Figura 2.24: Pila protocolos *BLE mesh*. Fuente *Bluetooth SIG, "Mesh Technology Overview"*

Bearer Layer

Esta capa, además de definir cómo se intercambian los mensajes entre los nodos que constituyen la red, sirve como interfaz para conectar las capas superiores de la pila *Bluetooth Mesh* y la especificación *Bluetooth Low Energy*. En ella se definen por tanto dos portadoras, *advertising bearer* y *GATT bearer*. La *advertising bearer* es la portadora principal y aprovecha las funciones de publicidad y escaneo GAP BLE para transmitir y recibir paquetes. Por otro lado, la *GATT bearer* permite que un dispositivo que no es compatible con la *advertising bearer* se comunique indirectamente con los nodos de una red de malla que sí lo hacen, es decir, los nodos *proxy*.

Network Layer

La capa de red se encarga principalmente de definir cómo se dirigen los diversos tipos de mensajes hacia uno o más elementos y de decidir si se aceptan o rechazan estos mensajes. También define el formato de mensaje de red que permite que las PDU(*Protocol Data Unit*) de la capa de transporte sean transportadas por la capa portadora o *Bearer Layer*. La capacidad de los nodos relays viene definido en esta capa.

Lower Transport Layer

La capa inferior de transporte coge las PDUs de la capa superior de transporte y la envía a la capa inferior en un dispositivo emparejado. También realiza la segmentación y el reensamblado de PDUs. En

caso de que los paquetes PDUs sean muy grandes, la capa inferior los subdivide en múltiples PDUs. De este modo, en el otro dispositivo esta capa inferior los reensamblará y enviará a la capa superior.

Upper Transport Layer

Por otro lado, la capa superior de transporte es responsable del cifrado, descifrado y autenticación de los datos pasados de la aplicación a la capa de acceso. También es la responsable del control de la capa de transporte.

Access Layer

Esta capa es la responsable de definir cómo las aplicaciones usan la capa de transporte superior, es decir:

- Define el formato de los datos de aplicación.
- Define los procedimientos de cifrado, descifrado y autenticación llevados a cabo en la capa de transporte superior, verificando que han llegado procedentes de la red correcta.

Foundation models Layer

La capa del modelo de base es la responsable de la implementación de aquellos modelos relacionados con la configuración y gestión de toda red *mesh*.

Models Layer

Finalmente, la capa más alta de la pila de protocolos *BLE mesh* es la de modelos. Esta capa es la responsable de la implementación de los mensajes, estados y comportamientos específicos del sistema definidos en uno o más modelos, dando la funcionalidad a la red *mesh*.

2.4 *Bluetooth VS Zigbee*

Uno de los principales competidores de Bluetooth, sobre todo en el ámbito IoT y la domótica es *Zigbee*[9] dado las funcionalidades y características que ofrece. Asi, se convierte en una comunicación más simple y barata que su competidor además de ofrecer una mayor cantidad de dispositivos conectados(hasta 65535). Su funcionalidad es similar y ofrece diferentes tecnologías teniendo un amplio mercado desarrollado por marcas como *Xiaomi*. Entonces ahora la pregunta es, **¿Porque se ha elegido Bluetooth en su lugar?** Tras analizar e investigar ambos protocolos de comunicación se hizó una reflexión y una mirada de 360º no solo desde una perspectiva de presente, sino también de futuro. A día de hoy, el futuro reside en los dispositivos móviles cuyo mercado sigue en auge. Todos ellos implementan la funcionalidad de Bluetooth a diferencia que *Zigbee*, motivo por el que se ha decidido desarrollar la comunicación de este modo. Por tanto, se ha pensado que *Zigbee* no tiene tanto futuro al centrarse únicamente en la domótica.

Se ha realizado una comparativa de los diferentes protocolos de comunicación inalámbricas extraídas del artículo "*Features of Building MESH Networks Based on Bluetooth Low Energy 5.1 Technology* "[10]. Las conclusiones han sido:

- Respecto al alcance de transmisión de datos en relación a la potencia de salida, *BLE for mesh* y BLE ofrecen un mayor rendimiento.

Output power (dBm)	Data transmission distance, m			
	ZigBee	*BLE @ 1Mbps (mesh)*	*BLE @ 2Mbps*	*BLE @ 125kbps (long-range)*
0	196	248	238	-
+4	231	276	273	-
+8	280	345	333	756

Figura 2.25: *Zigbee vs BLE for mesh* - Alcance. Fuente IEEE

- La tasa de transferencia de BLE es bastante menor que la ofrecia por *Zigbee*. Sin embargo, es más estable puesto que mantiene mencionada tasa independientemente del número de saltos. A continuación se muestran los datos con un paquete de 100KB. Por este motivo, *BLE for mesh* es recomendable para el diseño de grandes redes.

Number of hops	Transfer rate [KB / s]		
	Thread	*Zigbee*	*Bluetooth Low Energy*
1	47	22	3
2	24	13	2
3	17	8	2
4	13	6	2
5	10	5	3
6	8	4	2

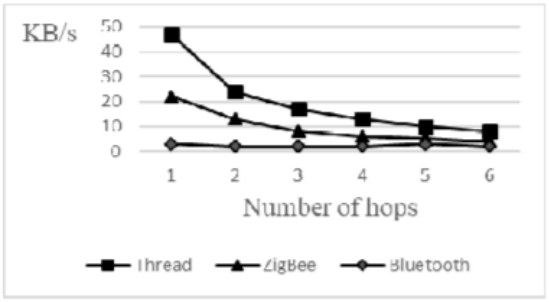

Figura 2.26: *Zigbee vs BLE for mesh* - Tasa de transferencia. Fuente IEEE

Entre otras ventajas que ofrece con respecto a *Zigbee*, su implementación es más sencilla y adaptable para dispositivos móviles. Además, las redes en malla Bluetooth pueden ser usadas para la detección y localización de dispositivos gracias a las últimas versiones del estándar, permitiendo construir un sistema de navegación y posicionamiento ya sea en pequeñas como en grandes áreas.

Por estos motivos y por su visión a futuro se consideró que era más óptimo el uso de Bluetooth en lugar de *Zigbee*.

Capítulo 3

MQTT

3.1 ¿En qué consiste?

En el capítulo anterior se ha estudiado en profundidad el estándar de comunicación Bluetooth y las importantes prestaciones que ofrece a la hora de elaborar un proyecto de tal calibre. Gracias a él, se pueden transmitir datos de sensores e interactuar con el HW de una manera simple.
Ahora la cuestión es **¿Podemos compartir estos datos con otros usuarios?**. La respuesta es sí. Para ello, se va a hablar del otro protocolo de comunicación usado en el proyecto, MQTT *(Message Queue Telemetry Transport)*[11][12]. Se trata de un protocolo de transporte de mensajes Cliente/Servidor basado en publicaciones/subscripciones. De este modo un usuario, **Cliente**, se suscribe a un servicio y cada vez que un mensaje es publicado, es recibido por el resto de clientes que estén suscritos a él. Todo este procedimiento se realiza a través de la nube. Se distinguen 2 elementos dentro de MQTT:

- **Clientes MQTT**
  Pueden publicar mensajes o recibirlos. Se encargan de realizar peticiones para solicitar datos a la nube (suscripción) o bien envian datos a la nube, ofreciendo su funcionalidad (publicación).
- **Broker/Servidor MQTT**
  Software que implementa el protocolo estableciendo la comunicación a nivel de aplicación. De este modo actúa como intermediario entre productores y consumidores de recursos. Además se encarga de autorizar el acceso e identificar los clientes (para averiguar si están suscritos a un servicio o no).

Un ejemplo del protocolo de comunicación MQTT se puede ver en la figura 3.1. En ella, un usuario a través de su ordenador se suscribe a un sensor que se encarga da publicar datos. Estos datos podrían ser por ejemplo procedentes de una estación meteorológica o de interés para un estudio acerca de la polución en un área como Madrid.

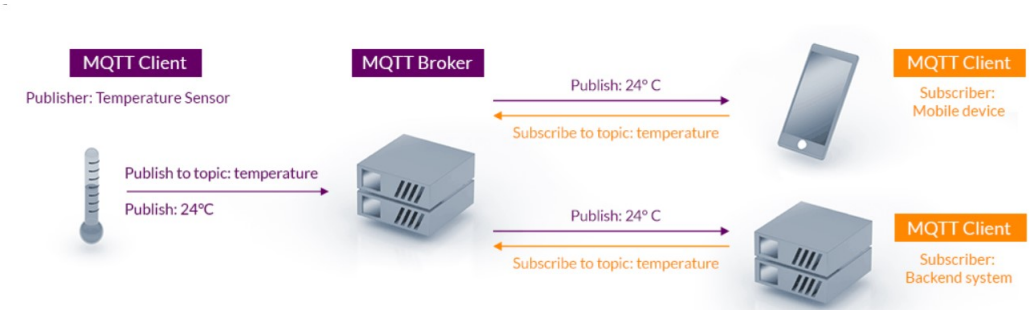

Figura 3.1: Ejemplo MQTT. Fuente mqtt.org

3.2 Características

Como se puede intuir, en efecto MQTT es de grandísima utilidad para el ámbito del Internet de las cosas. Su fortaleza reside en las prestaciones que nos ofrece:

- Funciona sobre TCP/IP.
- Ofrece un transporte de mensajes transparente y optimizado que reduce el tráfico en la red.
- Provee un mecanismo de notificación de conexiones inesperadas.
- No requiere mucho ancho de banda.
- No es necesario averiguar quien es el destinatario de los mensajes.
- Define 2 niveles de QoS
  - **Al menos una**
    Entrega segura pero con posibles duplicaciones.
  - **Exactamente una vez**
    Entrega segura de mensajes pero sin duplicaciones.

Dicho esto, es necesario mencionar que este protocolo dentro del gran conocido modelo OSI está ubicado entre las capas 5 y 7.

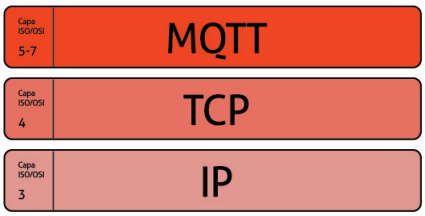

Figura 3.2: MQTT en el modelo OSI. Fuente códigoiot.com, "base de conocimiento MQTT"

3.3 Funcionamiento

Su funcionamiento se basa en un modelo **productor-consumidor** basado en publicaciones y suscripciones a temas (***topics***) en donde el *broker* es el encargado de gestionar el tránsito de mensajes por la nube. Los *topics* a los que esté suscrito un cliente se estructuran en una jerarquía de árbol, es decir se compone de varios niveles. Por defecto, MQTT opera sobre el puerto 1883 y el 8883 si funciona sobre TLS, es decir con seguridad sobre la capa de transporte.

Cuando un usuario desea realizar la conexión a un servidor, primero se conecta al intermediario por medio de un mensaje ***Connect*** que contiene nombre de usuario, contraseña, id. El broker acepta la petición con un mensaje ***CONNACK***

Figura 3.3: Conexión al broker. Fuente Luis Llamas, "¿Qué es mqtt? su importancia como protocolo IoT"

Posteriormente, si se ha aceptado la conexión el cliente ya puede suscribirse a un *topic*. Para ello, envía al broker un mensaje ***SUBSCRIBE*** que contiene la dirección alojada del servicio ofertado. El Broker responde con un mensaje ***SUBACK*** aceptando la suscripción

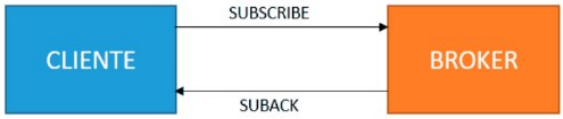

Figura 3.4: Suscripción MQTT. Fuente Luis Llamas, "¿Qué es mqtt? su importancia como protocolo IoT"

Por otro lado, si se desea realizar una publicación, un usuario envía un mensaje ***PUBLISH*** en el que reside la carga útil de datos. A diferencia que la suscripción, este mensaje no requiere un mensaje de confirmación del broker.

Figura 3.5: Publicación MQTT. Fuente Luis Llamas, "¿Qué es mqtt? su importancia como protocolo IoT"

De manera análoga a la suscripción de un *topic*, puede ocurrir lo contrario, es decir, que un usuario no desee recibir más datos. En este caso el cliente envía un mensaje ***UNSUBSCRIBE*** y el broker responde con un mensaje de confirmación ***UNSUBACK***.

En este proceso de intercambio de mensajes hay que mencionar que el cliente emite continuamente mensajes del tipo ***PINGREQ*** para comprobar si la conexión está activa. En caso de estarlo, recibe un mensaje ***PINGRESP***. Finalmente la desconexión se realiza con un mensaje ***DISCONNECT***.

El procedimiento completo del protocolo se muestra en la siguiente figura:

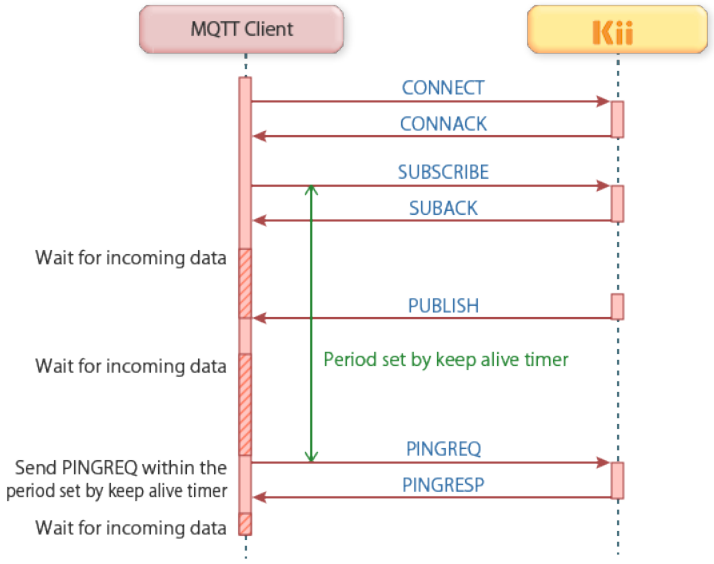

Figura 3.6: Procedimiento MQTT. Fuente mqtt.org

3.3.1 Formato de mensajes MQTT

En el proceso mostrado en la figura 3.6 se pueden observar todas las fases de una comunicación MQTT entre un cliente y un broker. Todo este procedimiento se realiza a partir de un conjunto de mensajes cuya estructura se divide en 3 partes; una serie de cabeceras de control y de tipado, cabeceras opcionales y la carga útil del mensaje, es decir, donde reside la información con un valor máximo de 256MB.

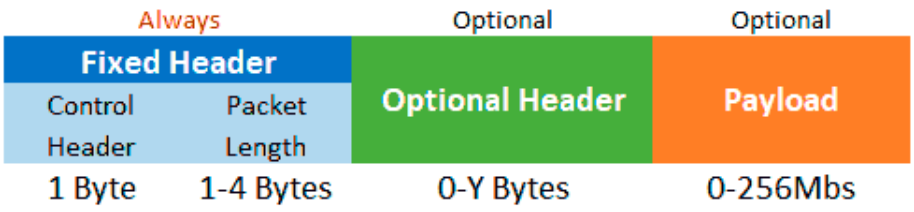

Figura 3.7: Formato mensajes MQTT. Fuente Luis Llamas, "¿Qué es mqtt? su importancia como protocolo IoT"

En la tabla siguiente puede observarse todos los valores posibles que pueden tomar las cabeceras MQTT con o sin QoS en una comunicación y en que sentido se envían los mensajes.

Nombre	Valor	Dirección	Descripción
Reserved	0	Forbidden	Reserved
CONNECT	1	Client=>Broker	Connection request
CONNACK	2	Broker=>Client	Connect ack
PUBLISH	3	Client <=>Broker	Publish message
PUBACK	4	Client <=>Broker	Publish ack (Qos 1)
PUBREC	5	Client <=>Broker	Publish received (Qos 2)
PUBREL	6	Client <=>Broker	Publish release (Qos 2)
PUBCOMP	7	Client <=>Broker	Publish complete(Qos 2)
SUBSCRIBE	8	Client =>Broker	Subscribe request
SUBACK	9	Broker=>Client	Subscribe ACK
UNSUBSCRIBE	10	Client=>Broker	Unsubscribe request
UNSUBACK	11	Broker=>Client	Unsubscribe ACK
PINGREQ	12	Client=>Broker	PING request
PINGRESP	13	Broker=>Client	PING response
DISCONNECT	14	Client<=>Broker	Disconnect notification

Tabla 3.1: Cabeceras MQTT. Fuente mqtt.org, "*MQTT specification*"

Capítulo 4

Hardware

En este apartado se pretende abordar todo lo referente al hardware usado en el proyecto. Todo el hardware usado es hardware libre, es decir, todas sus especificaciones son gratis y de acceso público para todo el mundo. Esto implica que el material usado sea más barato, pero manteniendo la función original, sin perder calidad. Por último, se pretende además, que todos los módulos utilicen estándares ya definidos sin tener que modificar nada con el fin de facilitar la implementación. A continuación, se pasa a explicar cada uno de los elementos que van a formar parte del proyecto.

4.1 Micro:bit

Micro:bit es uno de los microcontroladores elegidos para la elaboración del proyecto y será el encargado de constituir una parte de la red. Antes de entrar en detalles con sus especificaciones es necesario responder a la siguiente cuestión ***¿Qué es la micro:bit y qué relevancia tiene?***

La micro:bit[13] es una pequeña tarjeta programable cuyo objetivo final es acercar a las jóvenes generaciones a la programación de una manera divertida, fácil y que esté al alcance de todos. Nació para dar servicio en las escuelas británicas gracias a la BBC en 2012 junto con un apoyo gubernamental. Es ideal para niños de 11 años o más aunque esto es muy relativo ya que depende del interés y las ganas que pongan. A pesar de esto, y gracias a los sensores de los que dispone, puede utilizarse para IoT como es el caso de este proyecto. Aunque no tiene tanto rendimiento como otras placas muy conocidas como arduino, gracias a la calidad-precio que ofrece merece la pena ser probada.

La micro:bit dispone de un total de 3 versiones hasta el día de hoy pero para este proyecto se ha decidido utilizar la micro:bit V2 debido a la limitación de las anteriores versiones para este proyecto. Estas versiones son:

- **V1.3B**
- **V1.5**
- **V2**

El microcontrolador de micro:bit en su versión V2 está construido en base a un microprocesador *Nordic nRF52833* basado en un Cortex M4, junto con una matriz de leds 5x5, unos pines de entrada y salida además de alimentación, una serie de sensores, un conjuntos de módulos que utilizaremos para dar vida al proyecto, tres botones (A, B, logo) y un micrófono. Aunque por lo general se utiliza para la docencia, se puede usar también para IoT.

A continuación se muestra una imagen de la microbit V2 en la figuras 4.1 y 4.2:

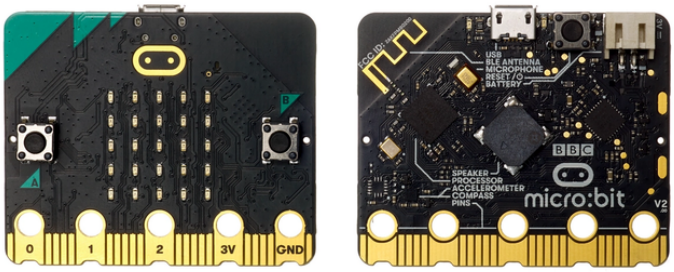

Figura 4.1: MicrobitV2. Fuente tech.microbit.org.

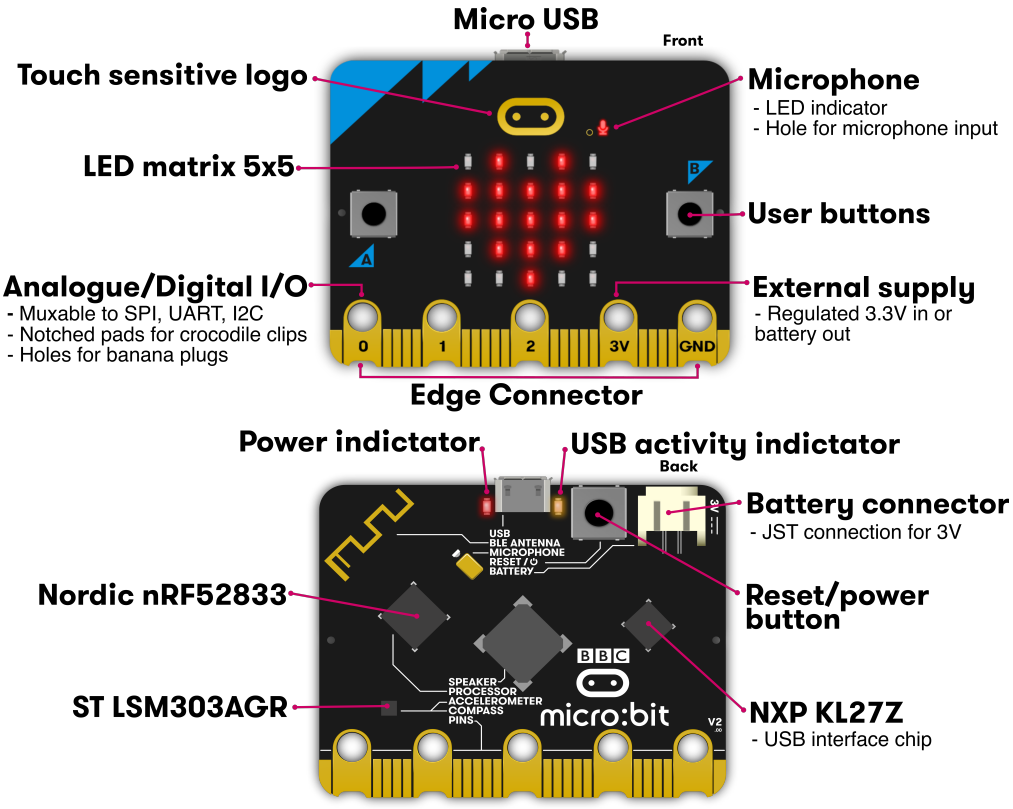

Figura 4.2: MicrobitV2 - detalle. Fuente tech.microbit.org.

Como se ha explicado anteriormente, para este proyecto en concreto se ha escogido la micro:bit en su versión 2. Esto es debido a que las anteriores versiones disponen de un procesador ya obsoleto para el entorno que se va a usar. Unido a esto, las versiones anteriores no soportan BLE *mesh*, necesario para la construcción del sistema IoT.

4.1.1 Especificaciones Hardware

Una vez explicada qué es la micro:bit y qué utilidades se le pueden dar, un punto necesario a mencionar es su especificación *hardware*[14]:

- Flash ROM de 512KB.
- RAM de 128KB.
- Velocidad 64 MHz.

En el Anexo 1 se muestra en total detalle el esquemático de la micro:bit V2 así como sus conexiones.

4.1.2 Bluetooth en la micro:bit V2

Un aspecto muy importante a tener en cuenta es el apartado Bluetooth, cuyas características más reseñables son las siguientes:

- *Stack Bluetooth 5.1 con Bluetooth Low Energy(BLE).*
- 50 canales de 2MHz, solo se usan 40 (0 to 39), 3 de *advertising* (37,38,39).
- Transmisioón de -40dBm a 4dBm
- Role Bluetooth como central o periférico.

Se debe de tener en cuenta que para poder trabajar con Blueooth en el entorno de desarrollo escogido es necesario la versión correcta de pila de protocolos Bluetooth que nos ofrece Nordic. Para la micro:bit V2 se debe de utilizar la versión S113 cuyas características más relevantes son [15]:

- Hasta 4 conexiones con periféricos y una *broadcaster* simultáneamente.
- Parámetros de conexión configurables.
- UUID personalizables.
- GATT Cliente y Servidor.
- GATT and GAP APIs.
- Capa de enlace soportada 1M PHY y 2M PHY.
- LE conexiones seguras.
- MTU configurable.

Un último punto a tener en cuenta en el HW de la micro:bit son la múltitud de sensores que nos ofrece. Aunque para este proyecto no se utiliza casi ninguno dado que en este caso no se les puede dar utilidad es necesario mencionarlos. Por este motivo, su utilización se centra en la configuración de los módulos procendentes del microprocesador de Nordic. En su versión V2 se disponen de los siguientes sensores y elementos auxiliares:

- Sensor de temperatura.
- Acelerómetro, brújula y magnetómetro.
- Altavoz.
- Un logo como botón táctil.

4.2 nRF52833DK

La otra tarjeta escogida para el diseño del sistema es la Nordic nRF52833DK. Se trata de una tarjeta cuyo procesador es el mismo que el usado en la micro:bit mencionada anteriormente, el nRF52833. Sin embargo, en este caso la placa ha sido desarrollada por la empresa de Nordic. La decisión de escoger esta tarejta ha sido por la documentación y utilidades que Nordic nos ofrece. Inicialmente se estudió la posibilidad de escoger otras como la nRF52840-dongle, sin embargo presentaba limitaciones a nivel de diseño. Por otro lado, también se planteó la posibilidad de usar la nRF5340DK pero su escasa documentación debido a su reciente salida ha hecho que su estudio e investigación se complicase.

Figura 4.3: nRF52840-dongle. Fuente Nordic Semiconductor

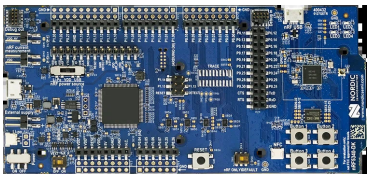

Figura 4.4: nRF5340DK. Fuente Nordic Semiconductor

Figura 4.5: nRF52833DK. Fuente Nordic Semiconductor

4.2.1 Especificaciones *Hardware*

Una vez presentado el microcontrolador mostrado en la figura 4.5 es hora de centrarse en el aspecto hardware y ver qué características hardware lo componen. Dado que tanto la micro:bit V2 como la nRF52833DK disponen del mismo procesador, comparten las mismas características no serán necesarias mencionarlas.

4.2.2 Módulos

A nivel de módulos, tanto la micro:bit como la nRF52833DK cuentan con los siguientes módulos más destacados al compartir el mismo procesador. La especificación viene presentada en su firmware[16]:

- 64 MHz ARM Cortex-M4.
- Soporta BLE y BLE *mesh* (con version S113 y S140).
- Soporta el protocolo de comunicación Zigbee.
- Soporta multiprotocolo.
- 12-bit ADC con ganancia configurable y 8 canales.
- UART, SPI, TWI, PDM, I2S, WDT.
- 4x4 canales PWM.
- NFC.
- Cristal oscilador de 32.768 KHz.
- 42 pines GPIO.
- 5 Timers y 3x RTC.
- Tensión configurable 1.7 - 5.5 V.

****Como anotación reseñable se remarca que no dispone de un DAC que por lo general se utiliza para generar señales de audio. Sin embargo, esto no presenta un problema ya que se puede simular su funcionamiento por medio de dos vías, una usando el módulo I2S y otro usando un canal PWM. En ambos casos es necesario tener las muestras almacenadas en un archivo para que posteriormente estos módulos las procesen.****

En el anexo 2, se muestra en detalle los pines y conexiones de los periféricos.

Como ya se ha hablado del Bluetooth previamente no es necesario mencionarlo. Sin embargo, dado que muchos de los módulos presentados se van a utilizar es necesario una explicación de los más importantes, en qué consisten y cómo funcionan con el fin de lograr un mejor entendimiento cuando se explique el desarrollo del sistema.

4.3 Explicación módulos

4.3.1 ADC

Los microcontroladores disponen de un ADC de hasta 12 bits con ganancia y tensión de referencia configurable y 8 canales. Se trata de un módulo cuya función es convertir señales analógicas en digitales. Al igual que en otros microcontroladores, el ADC puede trabajar en diferentes modos, destacando el modo *single shot*, y el modo *burst mode*. En el primer modo, únicamente muestrea un único canal y se para, mientras que en el segundo modo muestrea continuamente diferentes canales, consiguiendo de este modo datos de manera simultánea. Gracias al uso de módulos adicionales se puede lograr que esta conversión que realiza el ADC no sea bloqueante, es decir que el procesador no tenga esperas activas. En este caso, el ADC nos permite obtener datos de diferentes sensores como un sensor de distancia y un LDR entre otros. Se ofrece una imagen en la que se enseña el funcionamiento del ADC y como de una señal analógica se obtienen una mayor o menor cantidad de muestras según el tiempo de muestreo.

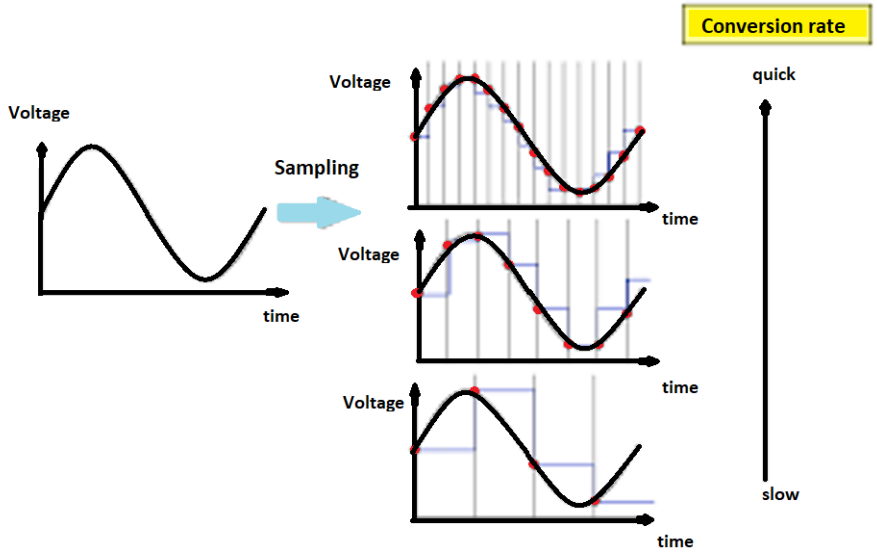

Figura 4.6: ADC

4.3.2 UART

Se trata de un protocolo de comunicación serie pero de manera asíncrona, es decir no tiene una señal de reloj para transmitir los datos. En cambio, es necesario que tanto el receptor como el transmisor trabajen a la misma velocidad de transmisión ya que sino se produce una mala interpretación de los datos. Las velocidades estándares más utilizadas son 9600, 38400 o 115200 bit por segundo (también llamado baudios). Por lo general se utiliza para la comunicación vía USB con la placa para cargar el programa y poder depurar el mismo por medio de un log de salida. Un ejemplo de como funciona la UART puede verse en la figura 4.7. Toda trama UART consta de un bit de *Start* a nivel bajo, entre 5 y 8 bits de datos y un bit de *Stop* a nivel alto, después si no se envían más datos, se mantiene en reposo a nivel alto. Opcionalmente puede tener 1 o 2 bits de paridad encargado de comprobar si los datos se han

recibido correctamente o no.

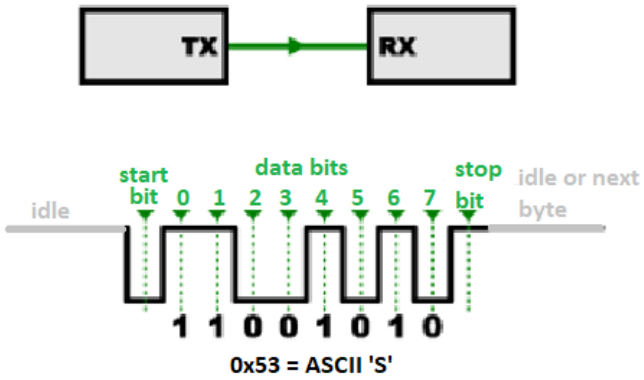

Figura 4.7: Trama UART

4.3.3 WDT

Se trata de un módulo de seguridad que normalmente se habilita y configura cuando un sistema cae en error. Consiste en un contador que cada cierto tiempo va decrementando de tal modo que cada vez que se ha ejecutado toda la funcionalidad del sistema se vuelve a realimentar. En caso de que no se dé esta situación, dos son las posibles explicaciones; el sistema ha tardado en procesar la funcionalidad más de lo necesario o ha caido en error. En estos casos, el sistema se reinicia con el objetivo de volver a la normalidad. A dia de hoy la mayoría de los sistemas empotrados lo disponen y el valor del contador se caracteriza por ser configurable

Se ofrece una figura en la que se muestra de una forma más visual el funcionamiento:

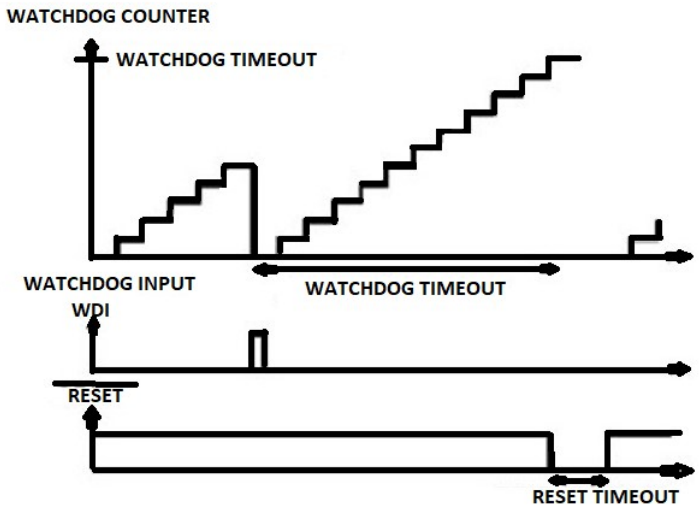

Figura 4.8: WDT

4.3.4 TWI (I2C)

TWI o también denominado I2C, se trata de un protocolo desarrollado para la comunicación serie síncrona de circuitos integrados. A diferencia de UART, explicado anteriormente, esta comunicación es síncrona, es decir, depende de una señal de reloj. Su principal ventaja reside en que el bus solamente consta de dos interfaces que permite hacer una comunicación de hasta 127 dispositivos.

Sus dos interfaces son el bus *Serial-Clock*(SCL), por el cual se envía la señal de reloj, utilizado para mantener la sincronía. Por otro lado está la interfaz *Serial-Data* (SDA), cuya misión es enviar los datos de forma bidireccional y sincronizados con el reloj proporcionado por el SCL.

Su arquitectura consiste en un maestro y uno o varios esclavos en el que la comunicación siempre es iniciada por el maestro quién pregunta previamente a los esclavos.

Respecto a la funcionalidad del envío de datos, la comunicación se inicia enviando una dirección de 7 bits del esclavo, y un bit de lectura o escritura según la acción que se quiera realizar. Como resultado el esclavo envía un bit de validación llamado ACK indicando que ha recibido la solicitud de comunicación y la acepta. Finalmente, se inicia la transmisión de datos en donde el bit de validación de nuevo estará presente y en este caso será enviado por el maestro o el esclavo según sea la acción de lectura o escritura. Para ilustrarlo mejor, se muestra un ejemplo de trama I2C en la figura 4.9. Cabe mencionar que para que la comunicación sea exitosa deben cumplirse las condiciones de *Start y Stop* remarcadas.

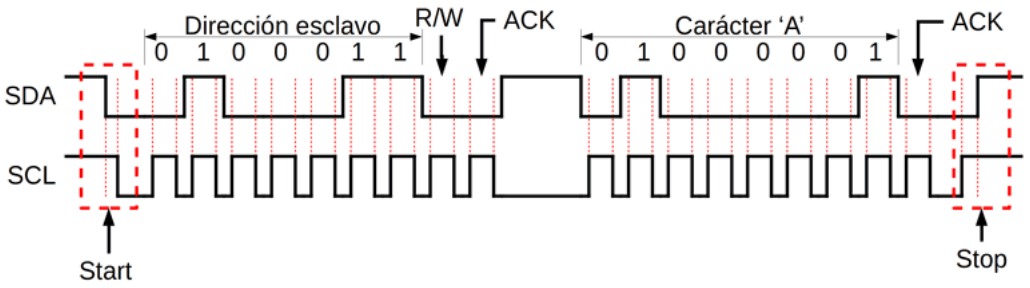

Figura 4.9: Ejemplo Trama I2C

4.3.5 PWM

Se trata de una modulación por ancho de pulsos cuyo objetivo consiste en modificar el ciclo de trabajo de una señal transmitida, modificando la cantidad de energía que se quiere transmitir a un dispositivo. Por este motivo, se trata de una modulación con un uso muy notable para controlar la luminosidad de una bombilla o controlar motores entre otros. Un ejemplo de lo mencionado puede observarse en la figura 4.10. En ella vemos la misma señal pero con diferentes ciclos de trabajo o energías que se transmite a un dispositivo que en este caso es un led. Se puede ver como a mayor ciclo de trabajo es decir, mayor tiempo está la señal a nivel alto, más intensa es la luz y viceversa. En un motor esto podría ser entendido como la velocidad del mismo. De esta manera y como se puede ver los usos son múltiples y su concepto es fácil de entender.

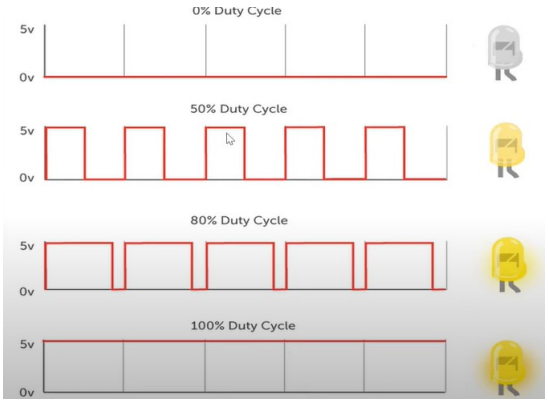

Figura 4.10: Ejemplo señales PWM

El nRF52833 dispone de 4 instancias PWM que contienen 4 canales PWM. Además cuenta con un total de 4 configuraciones cuyo uso depende de la casuística que se quiera tratar. Sin entrar en mucho detalle, estas configuraciones son:

- ***Common mode:*** Todos los canales de una instancia (4 canales por instancia) dan lugar a la misma salida.

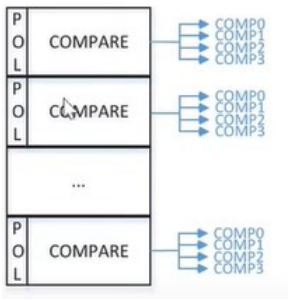

Figura 4.11: *Common mode.* Fuente Nordic Semiconductor, *"nRF52833 specification"*

- ***Group mode:*** Los canales de cada instancia se agrupan en dos salidas, una para los canales 0-1 y otra para los canales 2-3.

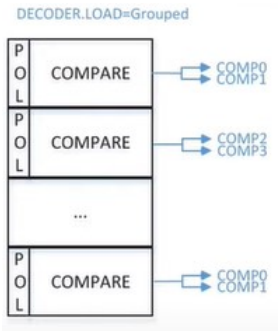

Figura 4.12: *Group mode.* Fuente Nordic Semiconductor, *"nRF52833 specification"*

- ***Individual mode:*** En cada instancia, cada canal tiene una salida que se corresponde con una secuencia determinada.

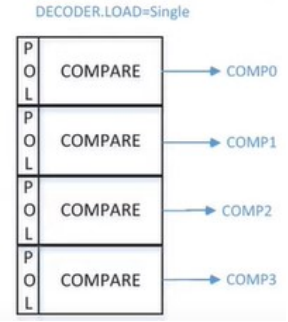

Figura 4.13: *Individual mode.* Fuente Nordic Semiconductor, *"nRF52833 specification"*

- ***WaveForm mode:*** En este modo el cuarto canal no puede ser usado ya que se utiliza como contador. Este modo suele utilizarse para desarrollar pwms personalizadas

.

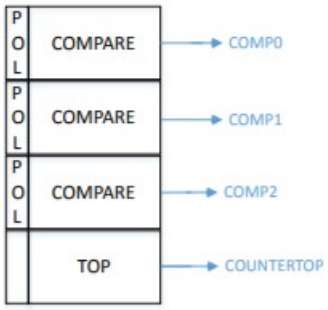

Figura 4.14: *WaveForm mode.* Fuente Nordic Semiconductor, *"nRF52833 specification"*

4.3.6 *Timers*

El nRF52833 cuenta con un total de 5 timers hardware además de la posibilidad de crear timers por sotfware que utilizan como base uno de los RTC. Los timers se tratan de uno de los módulos más importantes en todo sistema empotrado. En este proyecto su utilización se centra en el anunciamiento de datos y presencia de Bluetooth, concretamente la *Bearer layer*, mencionada en el capítulo 2, utiliza el Timer 2. Como dato adicional es necesario mencionar que el SoftDevice ya inicializa los RTC. Dos son los modos de funcionamiento de un timer; **modo match** y **modo capture**. En el modo match, los timers funcionan como contadores de tal modo que cuando la cuenta llega a un valor determinado de ticks, se reinicia la cuenta a 0 y se produce una interrupción. Por otro lado, en el modo capture los timers se utilizan para contar el número de ticks que han pasado entre flancos, permitiendo por ejemplo calcular el período de una señal.

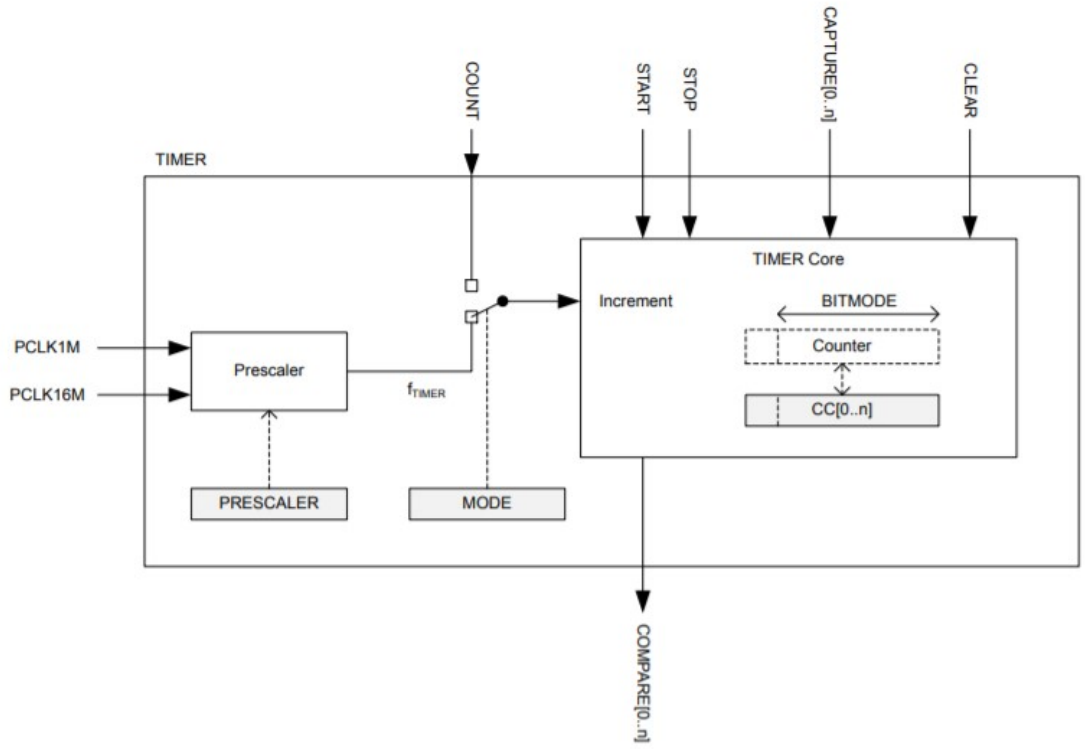

Figura 4.15: Arquitectura *timer*. Fuente Nordic Semiconductor, *"nRF52833 specification"*

Capítulo 5

Software

En este capitulo se va a hacer una descripción breve del Software para utilizar los elementos mencionados en el capítulo anterior. Como ya se ha mecionado anteriormente, tanto la micro:bit como la nRF52833DK cuenta con una amplia comunidad de desarrollo de software libre. Es necesario destacar el entorno de programación utilizado para desarrollar el código y el lenguaje propio de programación. De manera paralela también es necesario mencionar el entorno software utilizado para plasmar los datos del sistema. De este modo, se usan dos entornos diferenciados, uno para las placas y otro para plasmar los datos.

5.1 Entorno de programación de los microcontroladores

El entorno seleccionado para la programación de ambos tipos de microcontroladores se llama *Segger Embedded Studio(SES)[17]*. Es un IDE altamente recomendado cuya programación es en C para el desarrollo de aplicaciones que usan sistemas embedidos, teniendo como fiel competidor a Keil uVision usado durante el transcurso la carrera. Se trata de una aplicación de código abierto, es gratuita y se puede descargar para cualquier sistema operativo, tanto Windows, como MacOS y Linux. Esto permite que cualquier persona con un ordenador pueda empezar a programar. Así se convierte en un IDE muy potente que ofrece una gran versatibilidad para una gran cantidad de controladores, estando muy relacionado con los desarrollados por Nordic. De esta manera, se convierte en un IDE muy potente que tiene como lema el siguiente, “Cada byte cuenta“.

Numerosas son las herramientas con las que cuenta por lo que no es recomendado que sea usado para principiantes en la temática del desarrollo de sistemas embedidos. Posteriormente en el capítulo siguiente se mencionarán los aspectos necesarios para preparar el entorno y ser usado adecuadamente.

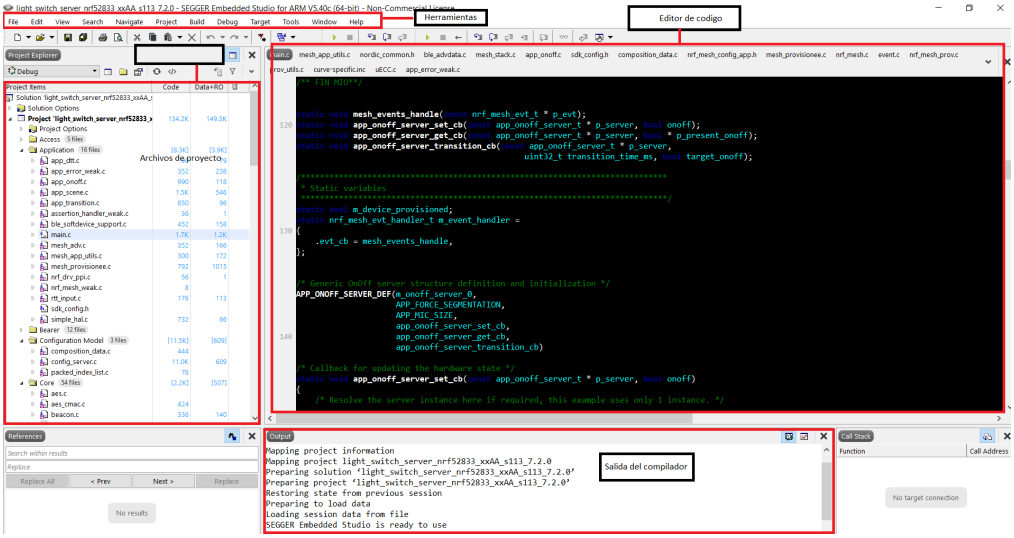

Figura 5.1: Vista del entorno SES

5.2 Entorno *Node-RED*

Por otro lado, el entorno escogido para la representación visual de los datos es *Node-RED[18]*. Caracterizado por su potencial y fácil utilización, se trata de una herramienta de desarrollo basada en flujos desarrollada para conectar dispositivos de hardware, API y servicios en línea como parte de IoT. Al igual que SES, Node-RED es *Open-source*, es decir, que puede instalarse en cualquier sistema operativo y cuya comunidad es ampliamente grande en donde todos pueden aportar y desarrollar ideas. La sencillez de aprendizaje y su uso hace que no sea necesario tener conocimientos de programación para ofrecer unas funcionalidades básicas, aunque si se desea se puede hacer una configuracion con lenguajes como *Javascript, JSON, HTML*. El aspecto visual se muestra a continuación con un ejemplo:

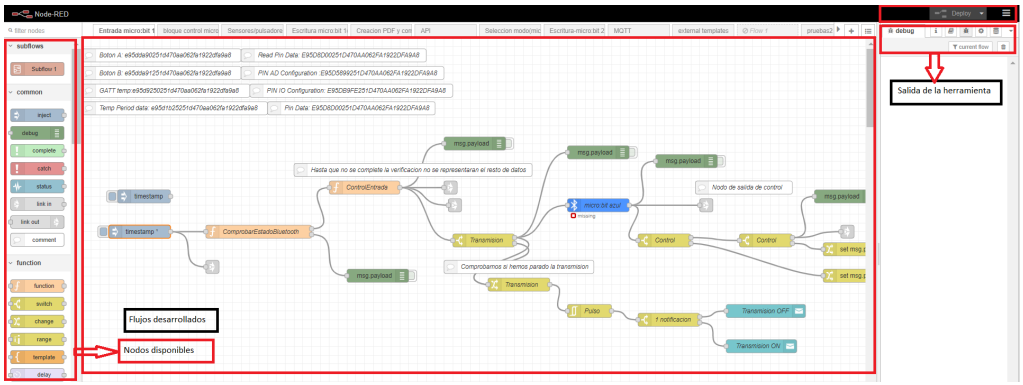

Figura 5.2: Vista del entorno Node-RED

Como se puede observar en la figura 5.2, Node-RED cuenta con un listado de nodos disponibles que vienen representados como cajas. Por defecto, ya vienen instalados un paquete de nodos pero pueden descargarse aun más. Posteriormente se explicará como hacerlo en el capítulo siguiente. Una vez se deseen utilizar los nodos basta con arrastrarlos a las plantillas y hacer las conexiones y configuraciones que se requieran. Por otro lado, mencionar que Node-RED nos proporciona un conjunto de herramientas que nos permite visualizar los datos, depurarlos entre otro tipo de personalizaciones.

5.3 Lenguajes de programación usados

En este apartado se habla de los lenguajes utilizados en los diferentes entornos seleccionados para la realización del proyecto, dando unas pequeñas pinceladas a como usarlos en los entornos según corresponda.

5.3.1 Segger Embedded Studio

El lenguaje utilizado para la programación de los microcontroladores en el entorno *Segger Embedded Studio* es el exitoso lenguaje C cuyo origen se remonta a los años 1970 gracias a los *Laboratorios Bell*. Es un lenguaje comunmente usado para la creación de sistemas y a partir de él han surgido muchas variaciones y extensiones del mismo como C++, creado a mediado de los años 1980.

A pesar de tener ya un largo recorrido, se sigue utilizando en diferentes ámbitos desde el mundo de la docencia hasta el mundo laboral, usándose como es el caso de este proyecto debido a qué es el lenguaje más eficiente en términos de velocidad de código y espacio despues del lenguaje ensamblador.

5.3.2 *Node-RED*

Varios son los lenguaje utilizados en el entorno de *Node-RED*. La utilización de ellos es opcional dado que los datos se transmiten en forma de flujo desde una entrada a una salida. Sin embargo, si se desea un tratamiento personalizado de los datos sí que será necesario. Entre los lenguajes más conocidos que soporta destacan:

- ***Javascript***: Uno de los lenguajes más utilizados a día de hoy gracias a las facilidades que ofrece y ampliamente usado en los navegadores web al ser interpretado, es decir, no requiere de una compilación como C, así como por su dinamismo. Su utilización es opcional en Node-RED si se desea una personalización de datos por medio de funciones u otros nodos. Este lenguaje suele estar unido junto con *JSON*, usado para describir datos.
- ***HTML* y *CSS***: Mundialmente conocidos, HTML y CSS se utilizan principalmente como lenguajes para el desarrollo de páginas web. Únicamente se utilizará estos lenguajes a la hora de representar los datos y no realizar un tratamiento de los mismos.

Capítulo 6

Preparación del entorno

En los capítulos 4 y 5 se ha introducido el hardware que se va a utilizar en el proyecto así como las herramientas Software que van a ser necesarias. Sin embargo, estas herramientas requieren no solo de una instalación sino también de una configuración. Por ello, en este capítulo se pretenden abordar esos puntos con el fin de ahorrar futuros problemas. Con el fin de seguir un orden lógico a como se hizo en el capítulo 5 se va explicar primero la instalación y configuración de *Segger Embedded Studio* y posteriormente de *Node-RED*.

6.1 Instalación y configuración de *Segger Embedded Studio*

La instalación de *Segger Embedded Studio* requiere de realizar una descarga. Por tanto, primero será necesario encontrar el instalador al que se puede acceder en la ruta siguiente `https://www.segger.com/downloads/embedded-studio/`. En él se puede escoger el sistema operativo que se desee.

Una vez se tenga descargado e instalado, saltará una alerta para activar una licencia. Por suerte, no será necesario disponer de una para poder utilizar el IDE.

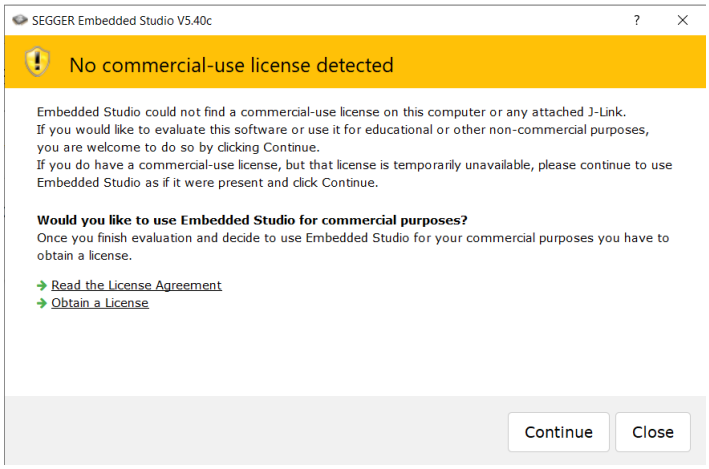

Figura 6.1: Mensaje de alerta de licencia en SES

Una vez realizada la instalación, para poder utilizar el entorno en los microcontroladores se requiere de descargar el SDK propio de Nordic. Dos son los SDKs necesarios para conseguir el funcionamiento del sistema.

- nRF5 SDK.
  Disponible en `https://www.nordicsemi.com/Products/Development-software/nrf5-sdk`.
- nRF5 SDK for Mesh.
  Disponible en `https://www.nordicsemi.com/Products/Development-software/nrf5-sdk-for-mesh`.

El primero engloba el funcionamiento de todos los módulos mencionados con anterioridad, mientras que el segundo contiene las librerías necesarias para utilizar *BLE mesh*. Tras descargar ambos SDKs, el siguiente paso es descomprimirlos y ubicarlos en el disco local C: del siguiente modo:

Figura 6.2: Carpetas SDKs

En estas carpetas hay una gran cantidad de ficheros con las diferentes librerías y códigos. También hay una cantidad muy generosa de ejemplos con los que se puede ir aprendiendo a utilizar los diferentes módulos mencionados en el capítulo 4 y otros.

Es de vital importancia comentar que para la correcta compilación y ejecución de los ejemplos de *BLE mesh* (que se encuentran en nRF5 SDK for Mesh, es necesario que el proyecto esté en la siguiente ruta del nRF5 SDK: *"C:/nRF5_SDK_17.0.2_d674dde/example/ble_peripheral"*.

6.1.1 Adaptar la configuración del compilador con los microcontroladores

- 1. Click derecho en la solución e ir a *options*.
- 2. Cambiar el despegable y de todas las configuraciones ir a *Common*. Luego ir a *Linker* en donde se hace configuración de la memoria.

En este punto la compilación puede ser correcta. Sin embargo, el sistema puede dar fallos a la hora de cargar el programa.

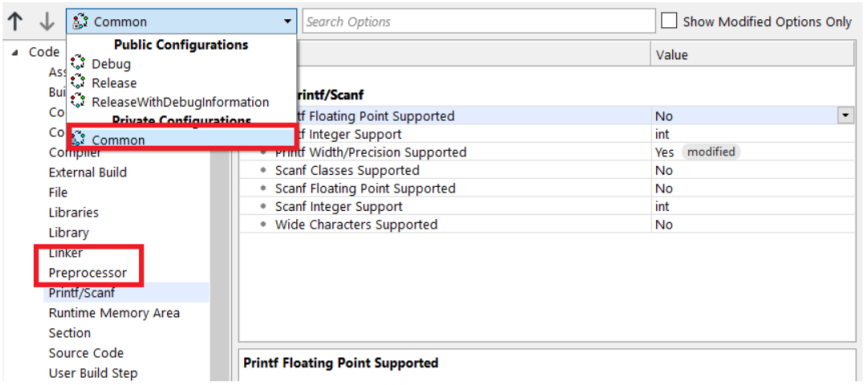

Figura 6.3: Paso 2

- 3. En *Linker* ir a *Memory Segments, Section Placement Files y Section Placement Macros* que se verá en los puntos siguientes.

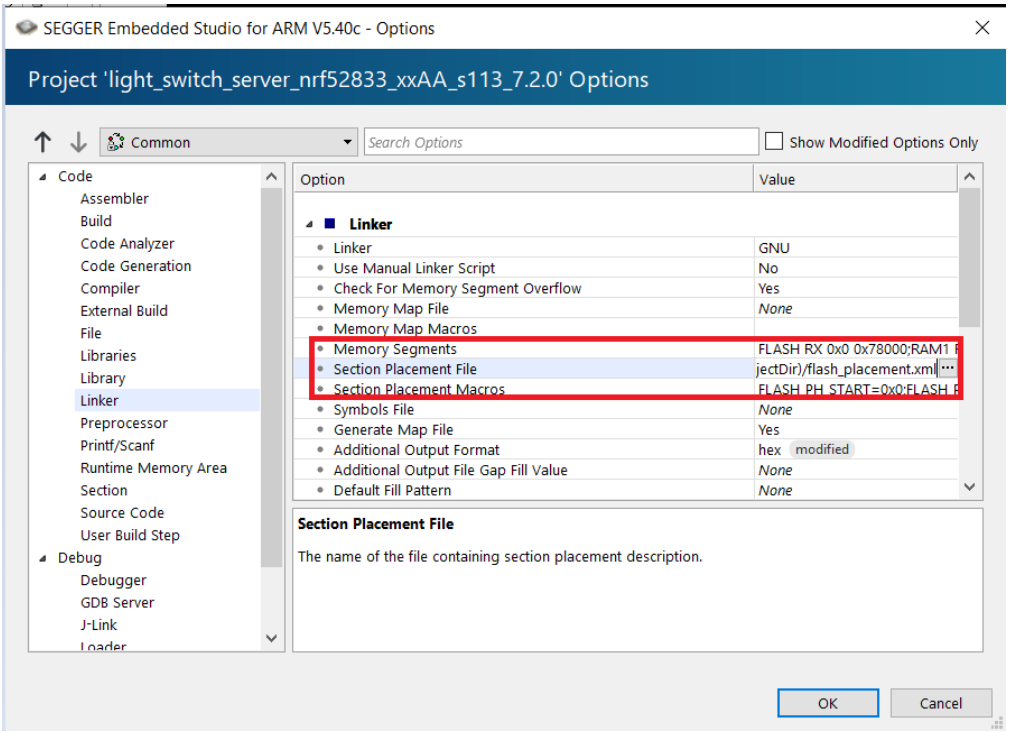

Figura 6.4: Paso 3

- 4. Pegar los valores de las figuras 6.5 y 6.6

Como nota indicar que estos valores son los indicados en el capítulo 4 procedentes de las características de memoria RAM y Flash. Al tener el mismo procesador, la configuración para ambos tipos de microcontroladores es el mismo.

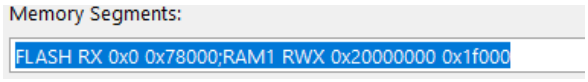

Figura 6.5: Valores Memory Segments

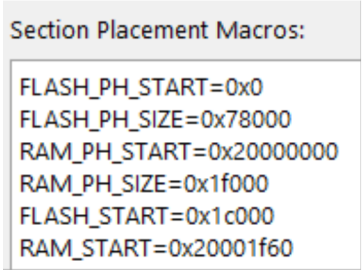

Figura 6.6: Section Placement Macros

- 5. Abrir el fichero *Section Placement Files.xml* y pegar el valor de las líneas indicas en la figura 6.7. Si este paso no se hace dará fallo de compilación.

```
35    </MemorySegment>
36    <MemorySegment name="RAM1" start="$(RAM_PH_START)" size="$(RAM_PH_SIZE)">
```

Figura 6.7: Modificación Section Placement Files

- 6. Finalmente ir a *Preprocessor* señalado en la figura 6.3 y pegar los valores de la figura siguiente en donde aparece la versión del controlador, la librería a la que se hace referencia (pca10100), la versión de la pila de protocolos S113 y una constante de procesador marcada cuyo valor varía según el procesador usado (MICROBIT o PLACA_GRANDE).

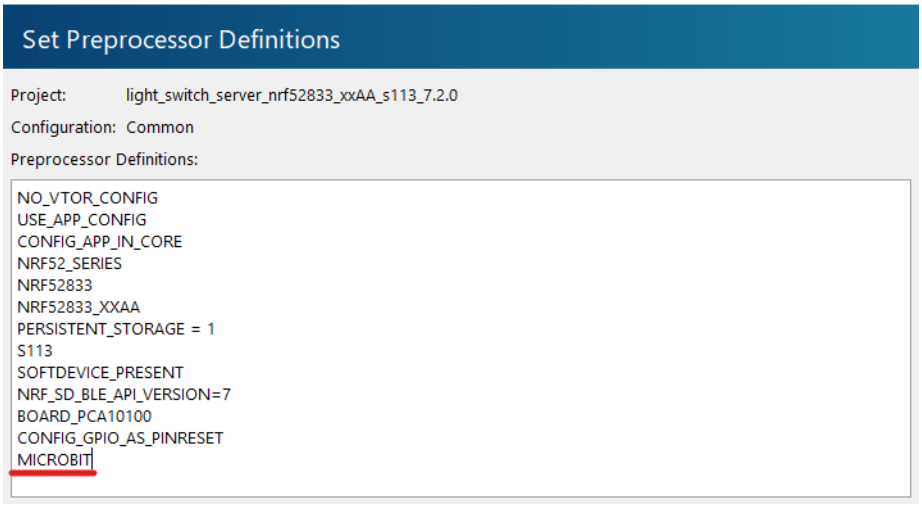

Figura 6.8: Preprocessor

6.1.2 Cambiar *firmware* micro:bit

Como último punto hay que mencionar que mientras que para la nordic nRF52833DK no es necesario, para la micro:bit V2 hay que instalar un *firmware* que proporciona SES con el fin de que el código funcioné correctamente al utilizar este entorno. El *firmware* se puede descargar desde la página de SES en el siguiente enlace `https://www.segger.com//downloads/jlink#BBC_microbit`[19]

Una vez descargado para instalarlo en la micro:bit hay que llevarla al modo *reboot*, pulsando inicialmente el botón de *Reset* antes de alimentarlo y luego alimentarlo. Finalmente basta con arrastrar el archivo en la unidad *MAINTENANCE* que se muestra a continuación

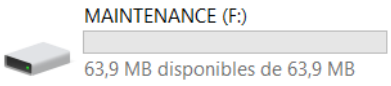

Figura 6.9: Modo *reboot* en micro:bit

6.2 Instalación y configuración *Node-RED*

La instalación de Node-RED al igual que la de *Segger Embedded Studio* es bastante sencilla y puede realizarse en cualquier sistema operativo al ser una plataforma *Open Source*. Sin embargo, en este caso la instalación se hará vía comandos a través del terminal o CMD[20]. Comentar que se ha elegido como sistema operativo *Windows* dado que en *Linux* hay ciertas librerías que daban problemas de compatibilidad. Por tanto, tras hacer este análisis se optó por esta opción. Para realizar una instalación correcta hay que seguir los siguientes pasos:

- 1. Antes de instalar Node-RED es necesario instalar una serie de dependencias como *NodeJS*[21]. A la hora de instalarlo debemos de seleccionar la opción de *chocolatey*. Con ello aparecerá una ventana de PowerShell que instalará más dependencias.

- 2. Una vez instalado, ya se puede usar el comando *npm* habilitado para poder instalar Node-RED. Por tanto, ejecutamos el comando *npm install -g –unsafe-perm node-red*.

Hecho esto, ya se puede abrir Node-RED ejecutando simplemente el comando *node-red*. Automáticamente aparece el siguiente mensaje mostrado en la figura 6.10, indicando que para abrir Node-RED hay que ir a la dirección localhost *127.0.0.1:1880*.

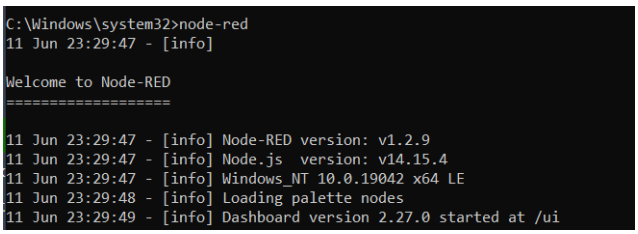

Figura 6.10: Como iniciar Node-RED

El siguiente punto es aprender como instalar librerías desde el entorno entre las que destacan las bibliotecas *dashboard y BLE*.

- 3. En *Node-RED* ir a *Manage-palettes* como se muestra en la figura 6.11

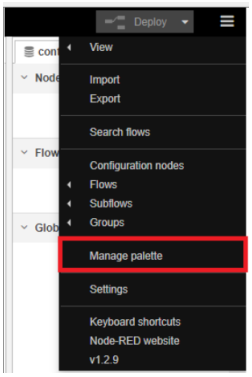

Figura 6.11: Acceder a las librerías

- 4. Instalar la biblioteca dashboard, buscando por node-red-dashboard. Finalmente para acceder a la dashboard basta con ir a la dirección *127.0.0.1:1880/ui*

Gran cantidad de bibliotecas pueden ser encontradas en la página oficial de Node-RED, muchas de ellas desarrolladas por usuarios. Con ellas se puede incluso extraer datos y convertirlos en informes, utilizar APIs entre otras posibilidades. Por otro lado, la instalación de la biblioteca Bluetooth es bastante más complicada. Esto es debido a que se requiere instalar una serie de dependencias adicionales y elegir la opción correcta. Además se ha necesitado utilizar un adaptador Bluetooth 4.0 y una herramienta adicional llamada *Zadig* cuya misión es cargar los drivers de Bluetooth en el adaptador. Por este motivo, se va a explicar paso a paso como hacerlo y así evitar quebraderos de cabeza innecesarios y pasar más tiempo del necesario investigando.

6.2.1 Instalación de librerías Bluetooth en *Node-RED*

- 1. En el terminal, escribir el comando *npm install –g –unsafe-perm node-gyp.*
- 2. Ir al directorio donde se tienen instalado node red, por ejemplo: *C:/Users/aleja/.node-red* y modificar el archivo *binding.gyp* con lo siguiente[22]:

{ "targets": [{ "target_name": "binding", "sources": ["build/Release/binding.node"] }] }

- 3. Posteriormente ejecutar los comandos *node-gyp configure y node-gyp build.* Si todo sale bien saldrá un *node-gyp: OK* como resultado.
- 4. Instaladas las dependencias, ahora toca instalar la librería BLE. Para ello ejecutamos el comando *: npm install –g –unsafe-perm node-red-contrib-generic-ble*[23].
- 5. Si se ejecuta de nuevo el entorno Node-RED saldrá el siguiente error:

```
27 May 15:02:58 - Error: No compatible USB Bluetooth 4.0 device found!
    at BluetoothHciSocket.bindUser (C:\Users\Loick\.node-red\node_modules\@abandonware\bluetooth-hci-socket\lib\usb.js:9
1:11)
    at BluetoothHciSocket.bindRaw (C:\Users\Loick\.node-red\node_modules\@abandonware\bluetooth-hci-socket\lib\usb.js:47
:8)
    at Hci.init (C:\Users\Loick\.node-red\node_modules\@abandonware\noble\lib\hci-socket\hci.js:120:18)
    at NobleBindings.init (C:\Users\Loick\.node-red\node_modules\@abandonware\noble\lib\hci-socket\bindings.js:93:13)
    at C:\Users\Loick\.node-red\node_modules\@abandonware\noble\lib\noble.js:61:24
    at processTicksAndRejections (internal/process/task_queues.js:75:11)
```

Figura 6.12: Error librería BLE Node-RED

- 6. Para resolver este problema es necesario tener un adaptador Bluetooth4.0 y la descarga de *Zadig*[24].
- 7. Al abrirlo saldrá una ventana como la representada en la figura 6.13. Pulsar en *Options ->List All Devices*, seleccionar el adaptador e instalar el driver.

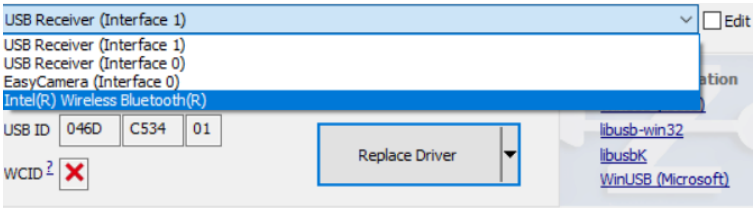

Figura 6.13: Zadig

- 8. El último paso es vincular los IDs del adaptador (indicados por Zadig en la figura 6.13) a nuestra librería de Node-Red. Para ello hay que ir a la ruta siguiente C:/Users/aleja/.node-red/node_modules /@abandonware/bluetooth- hci-sockets/usb.js y modificar el archivo js con los IDs mencionados.

```
this. usbDevice = usb.findByIds(0x8087,0x0AAA);
```

Figura 6.14: Zadig IDs parte 1

```
{vid: 0x8087, pid: 0x0AAA },
```

Figura 6.15: Zadig IDs parte 2

Capítulo 7

Desarrollo del sistema

Una vez adquiridos todos los conceptos teóricos y explicados qué elementos HW y SW se requieren, es hora de pasar a la parte práctica del proyecto en la que se aplican los conocimientos explicados. En primer lugar se va a hacer un resumen a grosso modo de los conocimientos adquiridos que se van a usar:

- Con el **análisis y estudio de los estándares de comunicación Bluetooth y sus variantes, BLE y *BLE for mesh* (capítulo 2**), se ha aprendido en qué consiste Bluetooth, y la posibilidad de crear servicios y características personalizadas además de comunicar microcontroladores entre sí a través de la red en malla.

- Con el **estudio del protocolo MQTT (capítulo 3)** y gracias al uso de un broker público se ha aprendido en qué consiste el protocolo y que posibilidades ofrece, pudiendo transmitir los datos de los servicios y características BLE desarrolladas hacia una página web, usando como *gateway* el PC.

- Con la **investigación y aprendizaje de los elementos HW y SW (capítulos 4 y 5)** se conocen el potencial que ofrecen y como diferentes módulos (de los cuales se han explicado algunos) permiten dar la funcionalidad deseada a los microcontroladores. Además se ha aprendido que se disponen de herramientas que permiten plasmar datos de una manera simple e intuitiva. Toda la programación se ha aprendido en base a la documentación ofrecida por nordic sobre el nRF52833 y los SDK[25][26].

- Y por último con **el capítulo 6** se ha aprendido como instalar los entornos a usar y su configuración, ahorrando un tiempo muy preciado para posibles futuros errores.

Con ellos, se consigue construir un sistema IoT robusto, práctico, de bajo coste y consumo que además permite no solo la monitorización de datos sino también una interacción con elementos Hardware de una manera cómoda y de lo más simple.

Dicho esto, es de vital importancia con el fin de organizar la explicación de este capítulo comentar cuales son los puntos que se van a tratar hasta llegar en el capítulo siguiente a unos resultados plausibles. Los puntos que se van a tratar en este capítulo son los siguientes:

- **Explicación del sistema.**
- **Desarrollo del sistema.**

7.1 Explicación del sistema

El primer punto a tratar en este capítulo pasa por explicar en qué consiste el sistema, dando un esquema de los elementos que lo componen y que va a ser capaz de hacer.
El sistema desarrollado se trata de una **red *mesh Bluetooth*** constituida por un total de 6 microcontroladores, 5 micro:bit V2 y una nordic nRF52833DK, tal y como se han presentado en el capítulo 4. Cada uno de estos nodos implementa un servicio BLE con el fin de monitorizar los datos de tal modo que por medio de los nodos con modelos clientes se puede actuar sobre los nodos modelos servidores, realizando cambios a nivel HW y SW. Finalmente, estos cambios, reflejados en los servicios BLE, se publicarán por el protocolo MQTT a un tópico gracias al uso de un broker público y por medio de Javascript y HTML se podrá representar estos datos al suscribirse a este tópico, realizando así la monitorización de datos. A continuación se representa un esquema de lo propuesto con el fin de dar un punto de vista más visual de lo mencionado en este punto, indicando los elementos que conforman la red así como que modelos implementan, la comunicación, sus IDs y como los datos son transmitidos por MQTT a la nube:

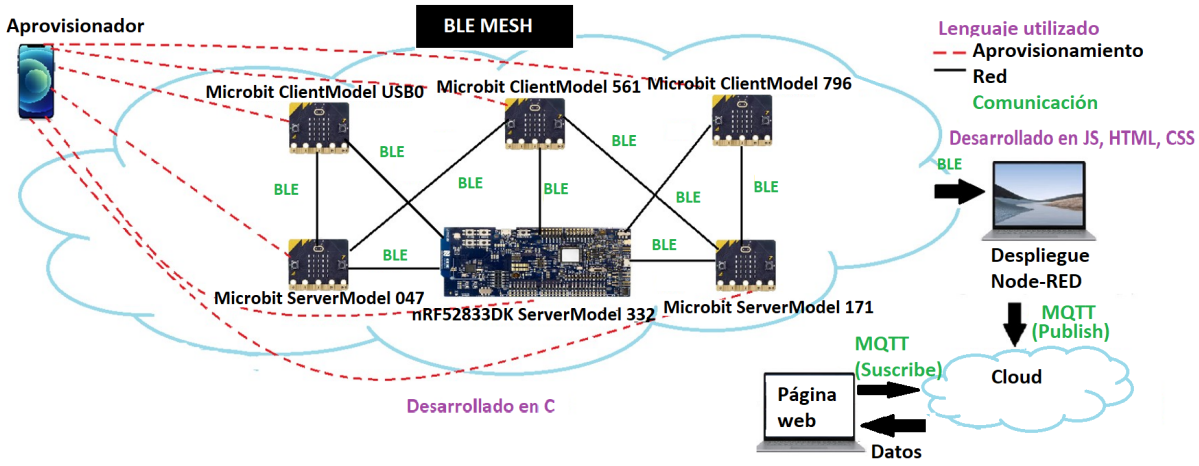

Figura 7.1: Sistema

En esta red, los nodos que la forman van a desempeñar uno ó más papeles o características *BLE mesh*:

- **La nRF52833DK va a implementar modelo servidor además de tener las características de *Proxy, Friend y Relay* propios de la red mesh.**
- **Las micro:bit ServerModel 047 y 171 van a implementar modelos servidores además de tener las características de *Proxy y Relay***
- **Las micro:bit ClientModel USB0, 561 y 796 van a implementar modelos clientes además de tener las características de *Low Power Node y Proxy*.**

Con ello, las micro:bit que implementan los modelos clientes se van a encargar de realizar peticiones a uno o más nodos que implementan modelos servidores (tal y muestra en la figura 7.1 por medio de direcciones que pueden ser unicast (dirigidas a un único nodo) o de grupo (dirigidas a un conjunto de nodos). A su vez actúan como **nodos *Relay***, retransmitiendo las solicitudes del cliente de tal modo que amplian el alcance del estándar *Bluetooth*. Por otro lado, la nRF52833DK además de cumplir con estos puntos, también va a ser un nodo amigo, permitiendo transmitir datos a los nodos *Low Power* cuando se encuentren dormidos.

Todos los nodos además de realizar su funcionalidad propia en la red van a implementar servicios y características BLE. Por ejemplo, la temperatura, el ciclo de trabajo de un servomotor entre otros. Gracias a esto, cada nodo puede representar el estado de sus elementos HW y valores de los sensores.

Estos valores así como los diferentes estados de los elementos alojados en características BLE son leídos por medio de ***Node-RED***, gracias a una librería BLE, y transmitidos por MQTT a un tópico. En este punto MQTT publicará los datos a un tópico y un *broker* se encarga de gestionarlos. Una vez publicados, un cliente externo que se suscriba al tópico representará los datos en una página web, permitiendo dar una visión dinámica de lo que está ocurriendo en el sistema.

7.1.1 Funcionalidad de la red *mesh*

En la figura anterior, se ha hecho una visión global de lo que sería la red *ble mesh* sin prestar atención a su funcionalidad y dando pinceladas de los elementos que lo componen. Esto se ha hecho así con el fin de no hacer muy complejo el esquema desde un primer momento y dar así una comprensión lineal del proyecto. Ahora es momento de profundizar, explicando la funcionalidad y qué elementos HW externos van a estar conectados a cada uno de los nodos, tomando siempre como referencia la figura 7.1.

Nodos con modelos clientes

Todas las micro:bit que implementan los modelos clientes van a tener dos modos de funcionamiento; modo configuración y modo funcionalidad. Se intercambia el uso de estos modos mediante el botón logo y visualmente se ve reflejado mediante un led rojo que se enciende si está en modo configuración o apagado si está en modo funcionamiento. En el modo configuración los nodos pueden iniciar/finalizar amistad con un nodo amigo (Botón A) o bien desaprovisionarse cuando haya problemas (Botón B).

- **micro:bit ClientModel USB0**
  Esta micro:bit va a implementar un total de **4 modelos; 1 modelo *dimming* y 3 modelos *onoff***. De esta manera, su funcionalidad va a ser la siguiente:
  Modelo 1 *dimming*: La micro:bit a partir de un potenciométro va a modificar el ciclo de trabajo de un led conectado a la *miro:bit ModelServer 047*.
  Modelo 2: Por medio del botón A se va a apagar o encender el led mencionado anteriormente.
  Modelo 3: A través de un sensor de gases, si el valor de este supera cierto umbral publicará un On haciendo sonar un zumbador conectado a la *micro:bit ModelServer 047*.
  Modelo 4: Por medio del botón B se va a mover un servomotor conectado a la *nRF52833DK (ServerModel 332)*.

Respecto a los servicios BLE, se van implementar un total de **2 servicios de lectura y notificación** que son de temperatura (gracias al sensor del que viene incorporado) y la cantidad de gas.
Se muestra un diagrama representativo de la conexión de los elementos con la micro:bit:

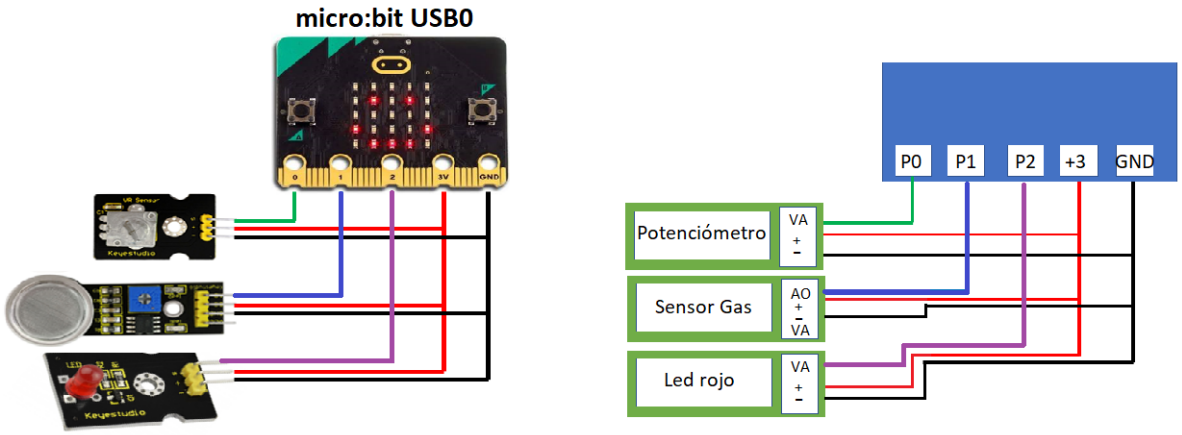

Figura 7.2: micro:bit clientModel USB0

- **micro:bit ClientModel 561**
  De todos los nodos, la micro:bit con el ID 561 es la más general. Esto es debido a que las publicaciones que va a realizar van a ser a una dirección de grupo a 3 nodos por medio de un único modelo cliente *onoff*. Con ello, si se pulsa el botón A enviará un ON, habilitando los módulos de los nodos y si se pulsa B enviará un OFF deshabilitándolos. En este caso, dado que solo tiene conectado un led al pin P2 no es necesario ofrecer un diagrama representativo.

- **micro:bit ClientModel 796**
  Por último, la micro:bit con el ID 796 al igual que las anteriores también va a implementar una serie de modelos cliente, en este caso 1 *dimming* y 2 *onoff*. A diferencia que los anteriores nodos, este va a configurar un total de dos modos de funcionamiento, manual y automático, para los elementos conectados a la nRF52833DK y micro:bit 171. De esta manera, en los modos automáticos los actuadores (servomotores y luz de los diferentes nodos) funcionan en base al valor de los sensores.
  Modelo 1: El modelo *dimming* se va a encargar de regular por medio de un potenciómetro una luz y un servomotor conectados a la micro:bit 171 (siempre y cuando esté en modo manual, de lo contrario no tendrá efecto).
  Modelo 2: Se encarga de que el ciclo de trabajo del servomotor y luz de la micro:bit 171 no esté en función del valor del potenciómetro, sino en función del LDR. Envía On u OFF si se pulsa el botón A.
  Modelo 3: Se encarga de que el servomotor conectado a la nRF52833DK se mueva en función del sensor de distancia y no del botón de la micro:bit USB0. Envía On u OFF si se pulsa el botón B.

En este nodo **las características BLE desarrolladas, de lectura y notificación, son para la temperatura y el valor de la luminosidad**.
Se muestra un diagrama representativo de la conexión de los elementos con la micro:bit:

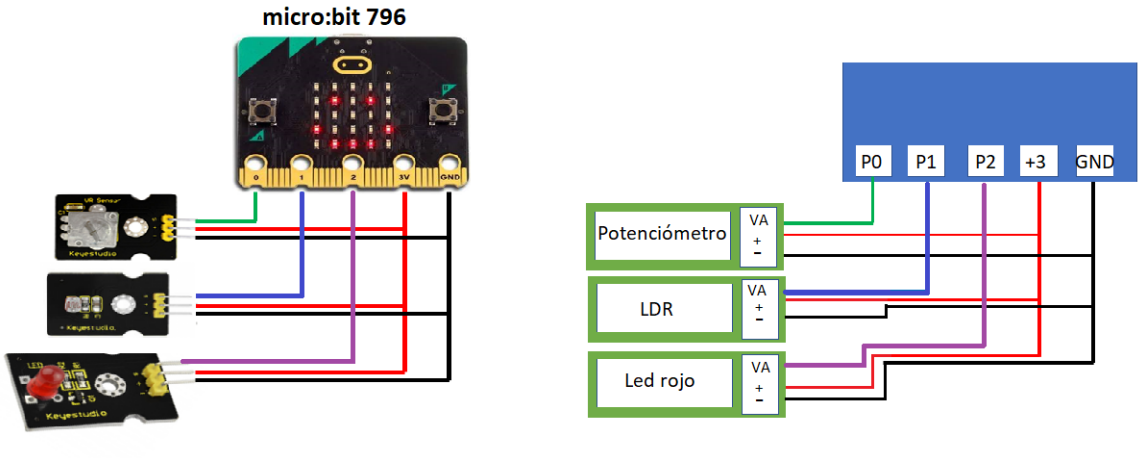

Figura 7.3: micro:bit ClientModel 796

Nodos con modelos servidores

Por otro lado, los nodos que implementan los modelos servidores van a ser los ubicados en la parte inferior de la figura 7.1. Todos ellos van a tener en común 2 servicios BLE, el estado del nodo (que indica si módulos como PWM, timer entre otros están habilitados) y el estado de la luz (que indica si está apagada o encendida).

- **micro:bit ServerModel 047**
  Este nodo implementa un total de 1 modelo *dimming* y 3 modelos servidor *onoff* cuyas funcionalidades están ligadas a los modelos clientes de la micro:bit USB0 y micro:bit 561. Estas son:
  Modelo 1: El potenciómetro regulado en el modelo cliente USB0 publicará el ciclo de trabajo que hace que aumente o disminuya la intensidad de un led.
  Modelo 2: Si recibe un ON u OFF de la micro:bit USB0, se enciende o apaga una luz.
  Modelo 3: Si el valor del sensor de gas de la micro:bit USB0 supera cierto umbral envía un ON que hace sonar un zumbador. Por el contrario cuando esté por debajo recibirá un OFF.
  Modelo 4: Si recibe un ON u OFF de la micro:bit 561, habilita o deshabilita módulos como PWM o timer entre otros.
  De esta manera, este nodo podría estar ubicado en la cocina de una casa. Las características desarrolladas, de lectura y notificación, son **estado del nodo, estado luz, intensidad de la luz (ciclo de trabajo)**.
  Se muestra un diagrama representativo de la conexión de los elementos con la micro:bit:

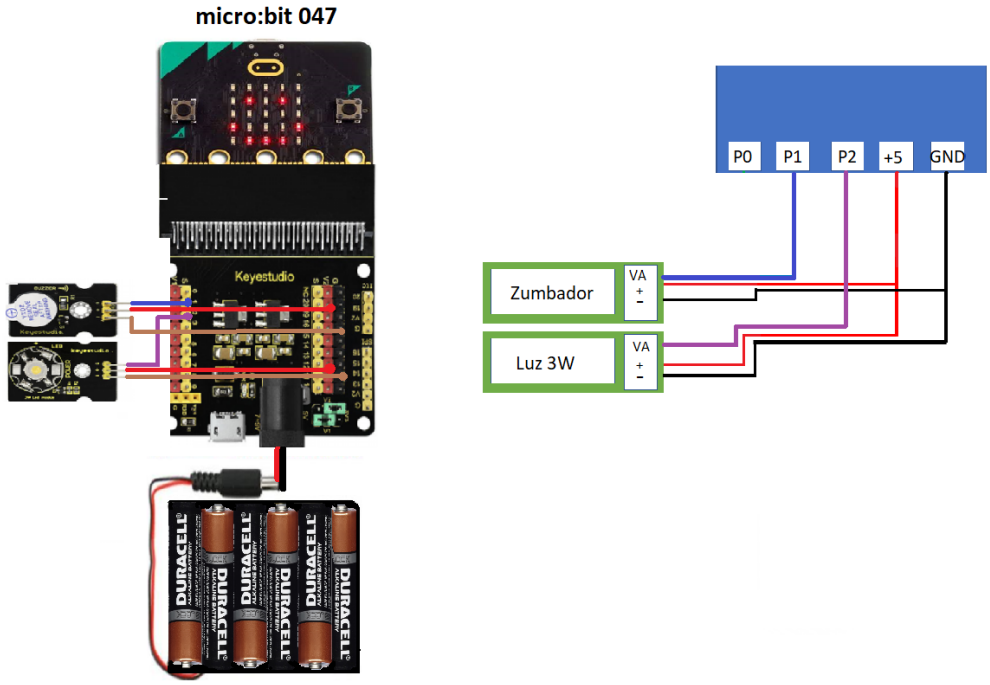

Figura 7.4: micro:bit ServerModel047

- **nRF52833DK Server Model 332**

  La nRF52833DK va a ofrecer un total de 3 modelos servidores *onoff* y van a operar gracias a los modelos clientes implementados en las micro:bit USB0, micro:bit 561 y micro:bit 796, correspondiendo un modelo por nodo. Gracias a ella es donde mejor se puede observar la funcionalidad de la *mesh* dado que recibe datos de hasta tres nodos. Además, este nodo va a implementar la característica *mesh* de nodo amigo, iniciando o finalizando amistad con los nodos que sean *Low Power*.

  La funcionalidad de esta placa como elemento de la *mesh* es:

  Modelo 1: Cuando recibe un On o un OFF de la micro:bit USB0 se mueve un servomotor, actuando como una puerta. Seguidamente al movimiento del servomotor suena un audio de "Puerta abierta" , "Puerta cerrada".

  Modelo 2: Este modelo es el encargado de cambiar la apertura de la puerta entre modo manual o automático. De este modo, cuando recibe un ON de la micro:bit 796 pasa a modo automático y lo que reciba en el modelo anterior es obviado dado que el servomotor se moverá en función del sensor de distancia. Seguidamente se reproduce un audio según la acción ocurrida.

  Modelo 3: Cuando recibe un On o un OFF de la micro:bit 561 se habilitan o deshabilitan los diferentes módulos configurados.

  Adicionalmente, la nRF52833DK tiene conectado un PIR y una luz de 3W de tal modo que cuando detecté presencia se encenderá la luz y arrancará un timer de 5s, posteriormente se apagará, simulando de esta manera la entrada de una casa.

  Las características, de lectura y notificación, que este microcontrolador ofrece son **cálculo de la distancia, estado de la luz, estado del nodo, estado puerta, modo puerta**. Se muestra un diagrama representativo de la conexión de los elementos con el microcontrolador:

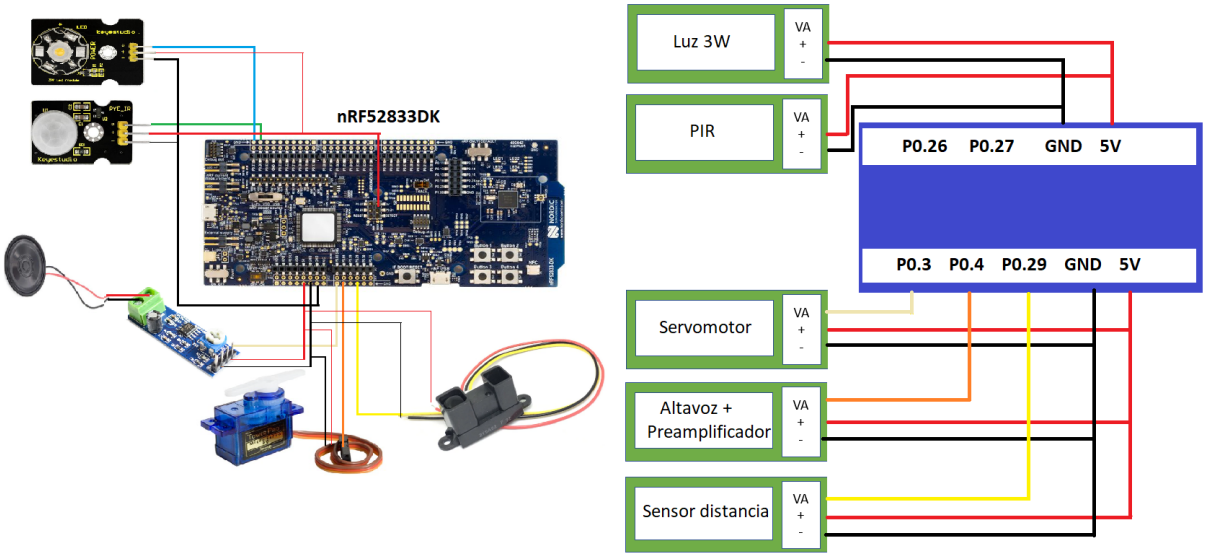

Figura 7.5: nRF52833DK ServerModel332

- **micro:bit ServerModel 171**

  Finalmente la micro:bit con el ID 171 al igual que las dos anteriores llevan a cabo una serie de modelos servidores con el fin de poder responder a las solicitudes de los modelos clientes. En concreto desarrolla 3 modelos; 1 *dimming* y 2 *onoff*, que se encargan de atender a las solicitudes realizadas por la micro:bit 561 y la micro:bit 796.

  La funcionalidades de estos modelos son:

  Modelo 1: Se trata de un modelo *dimming* servidor que recibe los valores enviados por el potenciómetro conectado a la micro:bit 796. De este modo, según el valor que reciba se modificará en mayor o menor medida una luz de 3W y un servomotor, actuando como si fueran unas hipotéticas persianas.

  Modelo 2: Se trata de un modelo *onoff* que responde a la petición onoff de un modelo de la 796. De este modo, cuando reciba ON se pasa a un modo automático en el que la luz y el servomotor varían en función de la luminosidad de un LDR conectado en el modelo cliente. En caso contrario, si se recibe un OFF de nuevo variarán según el potenciómetro.

  Modelo 3: Al igual que en los dos nodos anteriores, este modelo se suscribirá a una dirección de grupo de tal modo que cuando reciba ON u OFF de la micro:bit 561 se habilita o deshabilita una serie de módulos como por ejemplo de PWM.

  De esta manera, este nodo podría estar ubicado en el salón de una casa. Adicionalmente, se han desarrollado una serie de características de lectura y notificación como son **estado nodo, estado luz, ciclo de trabajo de las persianas**

  Se muestra un diagrama representativo de la conexión de los elementos con la micro:bit:

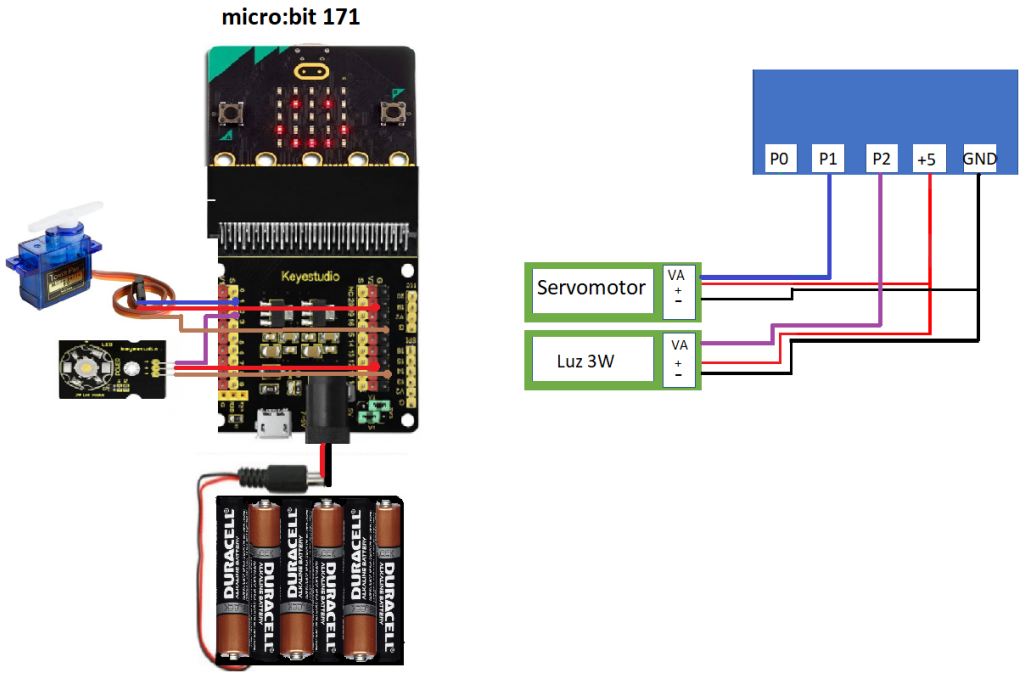

Figura 7.6: micro:bit ServerModel171

Con el fin de ofrecer un punto de vista más realista, se añade el montaje real de la red *mesh* representada anteriormente en la figura 7.1 junto con un posible uso real en una casa.

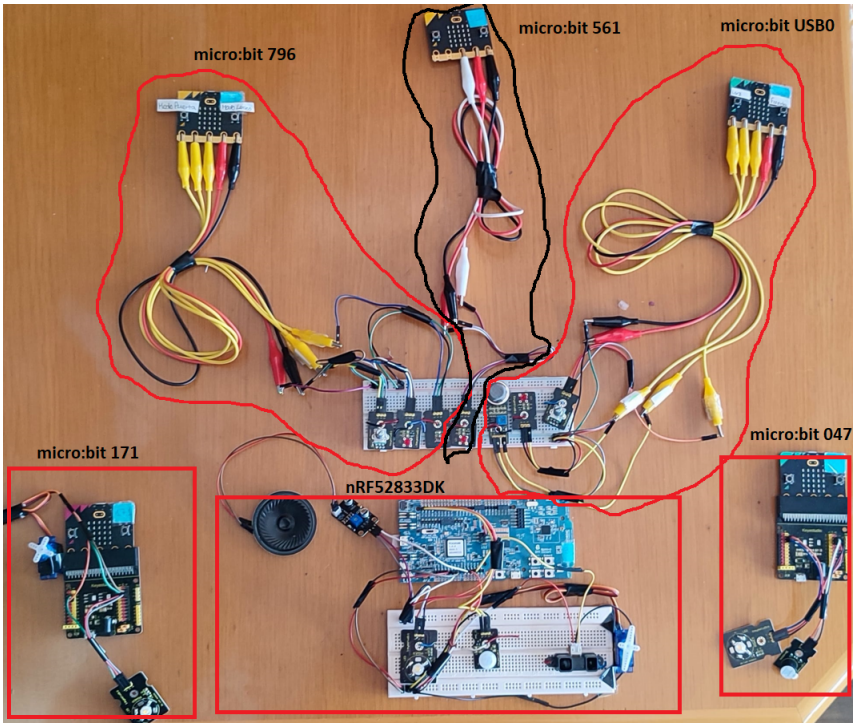

Figura 7.7: Montaje real del sistema

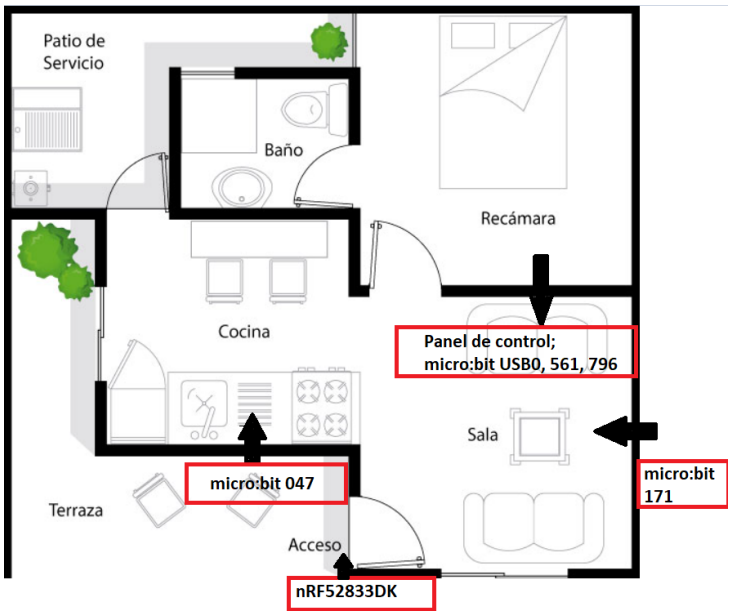

Figura 7.8: Uso real en casa

7.1.2 Servicios y características BLE

Como se ha ido indicando en el capítulo anterior, cada microcontrolador lleva impuesto un servicio BLE con varias características que permite al sistema monitorizar datos como la cantidad de luminosidad, la intensidad de luminosidad. Todas ellas son de lectura y notificación y con el uso de la red *mesh* se verán modificadas trabajando de manera paralela BLE y *BLE mesh*. Este punto tiene como fin aclarar que servicios y características se han planteado:

- **Temperatura**: Desarrollado en todos los nodos con modelos clientes. El valor es extraído a partir del sensor que incorporan los microcontroladores por medio de una API.
- **Estado nodo**: Desarrollado en todos los nodos con modelos servidores. Indica si los módulos están habilitados/inicializados o no.
- **Estado luz**: Desarrollado en todos los nodos con modelos servidores. Indica si la luz está encendida o apagada.
- **Modo luz**: Desarrollado en la micro:bit 171. Indica si la luz se regula por medio de un potenciómetro (manual) o a partir de un LDR (autómatico).
- **DutyCycle persianas**: Desarrollado en la micro:bit 171. Indica el ciclo de trabajo del servomotor que actúan como si fueran persianas. A mayor luminosidad, menor ciclo de trabajo(más subidas están).
- **Luminosidad**: Desarrollado en la micro:bit 796. Indica la cantidad de luminosidad recibida en el LDR. A mayor luminosidad menor dutyCycle envía al modelo servidor.
- **Estado puerta**: Desarrollado en la nRF52833DK. Indica si un servomotor que simula una puerta está abierta o cerrada.

- **Modo puerta**: Desarrollado en la nRF52833DK. Indica si la apertura de la puerta, es decir el movimiento del servomotor es manual (a partir del modelo cliente onoff implementado en la micro:bit USB0) o automática (a partir del sensor de distancia).
- **Distancia medida**: Desarrollado en la nRF52833DK. Indica la distancia medida por el sensor de distancia.
- **Intensidad luz**: Desarrollado en la micro:bit 047. Indica el ciclo de trabajo de la luz regulada por un potenciómetro (conectado a la micro:bit USB0).
- **Cantidad de gas**: Desarrollado en la micro:bit USB0. Indica la cantidad de gas en porcentaje que calcula en ese momento. Si supera cierto umbral envia un On a un modelo servidor de la micro:bit 047, haciendo sonar un zumbador.

7.2 Desarrollo del sistema

Explicado en qué va a consistir el sistema, que elementos lo componen y posibles aplicaciones, en este apartado se va a abordar su desarrollo, es decir como se ha codificado. Dado que la cantidad de librerías es bastante grande y el código elaborado es extenso al usar dos SDKs diferentes y haber un total de 6 códigos diferentes, se va a explicar únicamente las partes del código más relevantes y comunes que hacen posible el funcionamiento del sistema. A pesar de esto, el código va a subirse a un repositorio en donde estará disponible. En el capítulo 11 se encuentra el enlace para descargar todo el código fuente junto con una serie de anotaciones a tener en cuenta para el correcto funcionamiento.

7.2.1 Programación microcontroladores

La programación de los microcontroladores va a realizarse a partir del IDE mencionado en el capítulo 5, *Segger Embedded Studio*. Dado que se va a usar tanto BLE como *BLE for mesh* ha sido necesario la descarga de dos SDKs y por tanto trabajar con los dos a la vez, teniendo así en lo que nordic llaman, un *coexistance code*. Por otro lado, ha sido necesario modificar archivos de cabecera que ya venían de serie para adaptar el funcionamiento tanto a la nRF52833DK como a la micro:bit V2 al tener un pinaje diferente.
A nivel de código los aspectos más importantes a comentar son:

- **Configuración inicial**
  Inicialmente se configuran los módulos que vayan a ser necesarios. Esto se hace con las funciones ***initialize() y start()***

```
/*Llama a la configuracion de los diferentes modulos así como a los stack ble , ble for mesh y a los servicios que esten definidos*/
static void initialize(){
    bool erase_bonds;
    config_system(); /*configuramos pines como GPIO, inicializamos API de temperatura y timers SW necesario para BLE mesh y BLE*/
    NRF_LOG_INFO("Debug sobre UART y RTT iniciado.");

    __LOG_INIT(LOG_SRC_APP | LOG_SRC_FRIEND, LOG_LEVEL_DBG1, LOG_CALLBACK_DEFAULT);
    __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "----- Microbit Model Client 561 -----\n");

    buttons_leds_init(&erase_bonds);

    /*Se inicializan primero los modulos de BLE como su pila, parametros de conexion y perfil de acceso generico (el advertiser no e
    ya que BLE mesh cuenta con el bearer)*/
    ble_stack_init();
    gap_params_init();

    services_init();

    conn_params_init();

    //Se inicializa mesh
    mesh_init();

    ERROR_CHECK(sd_power_dcdc_mode_set(NRF_POWER_DCDC_ENABLE));

    //Se inicializa caracteristica LPN
    mesh_lpn_init();

}
```

Figura 7.9: *Initialize()*

La función ***initialize()*** inicaliza y configura módulos del SDK normal como son los pines como GPIO, los timers(***config_system()***), los botones como interrupción(***buttons_leds_init()***) y módulos relacionados con BLE como su pila de protocolos(***ble_stack_init()***), parámetros del GAP (***gap_params_init()***), parámetros de conexión(***conn_params_init()***) y el servicio BLE (***services_init()***). Por otro lado, se inicializa la pila de protocolos *BLE mesh*, los modelos de la *mesh* (***mesh_init()***) y la configuración del nodo. En este caso, el nodo al ser un modelo cliente lo configuramos como *Low Power* con ***mesh_lpn_init()***. El procedimiento que se ha hecho para los nodos con modelos servidores es el mismo, omitiendo este último paso al no ser LPN.

```
/*Arranca el sistema y el proceso de aprovisionamiento*/
static void start(void)
{
    //En caso de no estar aprovisionado se le pasa al proceso de aprovisionamiento la configuracion necesaria
    if (!m_device_provisioned)
    {
        static const uint8_t static_auth_data[NRF_MESH_KEY_SIZE] = STATIC_AUTH_DATA;
        mesh_provisionee_start_params_t prov_start_params =
        {
            .p_static_data    = static_auth_data,
            .prov_sd_ble_opt_set_cb = NULL,
            .prov_complete_cb = provisioning_complete_cb_aux,
            .prov_device_identification_start_cb = device_identification_start_cb_aux,
            .prov_device_identification_stop_cb = NULL,
            .prov_abort_cb = provisioning_aborted_cb_aux,
            .p_device_uri = EX_URI_LPN
        };
        //Arranca el aprovisionamiento
        ERROR_CHECK(mesh_provisionee_prov_start(&prov_start_params));
    }
    else
    {
        unicast_address_print();
    }

    mesh_app_uuid_print(nrf_mesh_configure_device_uuid_get());

    //Con todo hecho se arranca el stack mesh
    ERROR_CHECK(mesh_stack_start());

    //Se printea (necesario buggear) para saber lo que hace por si solo
    __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, funcionalidad_mesh);
}
```

Figura 7.10: *start()*

Posteriormente, configurados los parámetros de la *mesh* e inicializada su pila, en la función ***start()*** se prepara el proceso de aprovisionamiento, configurando los parámetros para el proceso así como una serie de funciones *call-back* que serán muy frecuentes en *BLE mesh*. Para todos los nodos, este paso también es común (con la única diferencia que en los nodos servidores se añade un handler para recuperar los valores del modelo). Hecho esto, el sistema ya está a la espera de eventos y se pone en bajo consumo con ***idle_state_handle()***, además de ejecutar la funcionalidad que le corresponda.

```
    for (;;)
    {
        (void)sd_app_evt_wait();
        idle_state_handle(); //Pasa el dispositivo a estado iddle reduciendo el consumo
        execFuncionality();

    }
}
```

Figura 7.11: Bucle while

- **Inicializar modelos Cliente**

  Todos los modelos se inicializan por medio de una función *call-back* pasada como parámetro en la función anteriormente mencionada ***mesh_init()***.

```c
/*Funcion encargada de inicializar el modulo mesh junto con la configuracion de los modelos, la pila de protocolos mesh...*/
static void mesh_init(void)
{
 //Para inicializar la pila de protocolos se requiere configurar los parametros como la priordiad(mas baja), los modelos, CLK y la config
    mesh_stack_init_params_t init_params =
    {
        .core.irq_priority       = NRF_MESH_IRQ_PRIORITY_LOWEST,
        .core.lfclksrc           = DEV_BOARD_LF_CLK_CFG,
        .core.p_uuid             = NULL,
        .models.models_init_cb   = models_init_cb,
        .models.config_server_cb = config_server_evt_cb
    };

    //Se inicializa la pila de protocolos mesh con los parametros configurados
    uint32_t status = mesh_stack_init(&init_params, &m_device_provisioned);
    switch (status)
    {
        case NRF_ERROR_INVALID_DATA:
            __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "Datos en memoria persistente estan corruptos. El dispositivo empieza desaprovisionado.\n"
            __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "Reboot antes de empezar con el aprovisionamiento.\n");
            break;
        case NRF_SUCCESS:
            break;
        default:
            ERROR_CHECK(status);
    }

    /*Añade al handler de la mesh el handler de amistad para averiguar si LPN ha recibido informacion de su amigo */
    nrf_mesh_evt_handler_add(&m_mesh_core_event_handler);
}
```

Figura 7.12: Parámetros *mesh*

En el caso del modelo cliente *onoff*, la inicialización se muestra a continuación. Pero el proceso es análogo para cualquier tipo de modelo cliente.

```c
/*Funcion call-back que llama a la inicializacion del modelo*/
static void models_init_cb(void)
{
    __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "Inicializando y agregando modelos al nodo\n");

    //Parametros del modelo
    ClienteOnOFF.settings.p_callbacks = &client_cbs;
    ClienteOnOFF.settings.timeout = 0;
    ClienteOnOFF.settings.force_segmented = false;
    ClienteOnOFF.settings.transmic_size = APP_MIC_SIZE;

    //Se inicializa el modelo
    ERROR_CHECK(generic_onoff_client_init(&ClienteOnOFF, 0));
}
```

Figura 7.13: Se inicializa el modelo

En esta función *call-back* se configuran otras funciones *call-back* que indican el estado del valor enviado, parámetros del modelo y posteriormente se inicializa el modelo, pasándole como argumento la estructura del modelo y el número del modelo.

- **Inicializar modelos Servidores**

  En el caso de los modelos servidores, el procedimiento será algo más elaborado. Esto es debido a que requieren de la definición de una instancia del modelo además de dos auxiliares para recuperar el valor del modelo en caso de haber algún problema como que el nodo se quede sin batería. La instanciación de los modelos servidor *dimming* y *onoff* recibe como parámetros el nombre de la instancia, funciones call-back de escritura(*set*), lectura(*get*), transición y parámetros opcionales como segmentación de datos. En la siguiente figura, se muestra como se realiza para el modelo *onoff*:

```
/*Definicion modelo servidor ONOFF*/
APP_ONOFF_SERVER_DEF(m_onoff_server_0, //nombre instancia
                     APP_FORCE_SEGMENTATION, //No fuerza segmentacion(false)
                     APP_MIC_SIZE, //Tamaño MIC
                     app_onoff_server_setLuz_cb, //call-back escritura
                     app_onoff_server_getLuz_cb, //callback-lectura
                     app_onoff_server_transitionLuz_cb); //transition
```

Figura 7.14: Instanciación modelo *onoff*

Mientras que para el modelo *dimming* se realiza así:

```
/* Definicion e inicializacion del modelo servidor dimming*/
APP_LEVEL_SERVER_DEF(m_level_server_0, //Instancia del modelo servidor
                     APP_FORCE_SEGMENTATION, //No fuerza segmentacion de mensajes mesh
                     APP_MIC_SIZE,  //Tamaño MIC
                     NULL,
                     app_level_server_set_cb, //cb escritura
                     app_level_server_get_cb, //cb lectura
                     app_level_server_transition_cb);  //cb transicion
```

Figura 7.15: Instanciación modelo *dimming*

Además de esta instancia, todo modelo servidor requiere de instancias adicionales que la acompañan para recuperar el valor en caso de que haya problemas de alimentación.

```
#if SCENE_SETUP_SERVER_INSTANCES_MAX > 0

APP_DTT_SERVER_DEF(m_dtt_server_0,
                   APP_FORCE_SEGMENTATION,
                   APP_MIC_SIZE,
                   NULL);

APP_SCENE_SETUP_SERVER_DEF(m_scene_server_0,
                           APP_FORCE_SEGMENTATION,
                           APP_MIC_SIZE,
                           app_level_scene_transition_cb,
                           &m_dtt_server_0.server);
#endif
```

Figura 7.16: Instancias extras

La inicialización es invocada como una función call-back dentro ***mesh_init()*** que llama a ***app_model_init()***. La inicializacion de los modelos servidores recibe como parámetros la instancia y el número de modelo:

```
static void app_model_init(void)
{
  //Modelo dimming servidor
  ERROR_CHECK(app_level_init(&m_level_server_0, APP_LEVEL_ELEMENT_INDEX));
  __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "App Level Model handle: %d\n",
        m_level_server_0.server.model_handle);

 #if SCENE_SETUP_SERVER_INSTANCES_MAX > 0
   /* Instantiate Generic Default Transition Time server as needed by Scene models */
   ERROR_CHECK(app_dtt_init(&m_dtt_server_0, APP_LEVEL_ELEMENT_INDEX));
   __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "App DTT Model handle: %d\n",
         m_dtt_server_0.server.model_handle);

  ERROR_CHECK(app_level_scene_context_set(&m_level_server_0, &m_scene_server_0));
   __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "App Scene Model handle: %d, Element index: %d\n",
         m_scene_server_0.scene_setup_server.model_handle,
         m_scene_server_0.scene_setup_server.settings.element_index);
#endif

  //Modelo OnOFF servidor 1 - Luz On OFF
  ERROR_CHECK(app_onoff_init(&m_onoff_server_0, APP_ONOFF_ELEMENT_INDEX));
  __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "App OnOff Model handle: %d\n",
        m_onoff_server_0.server.model_handle);

#if SCENE_SETUP_SERVER_INSTANCES_MAX > 0

   ERROR_CHECK(app_dtt_init(&m_dtt_server_1, APP_LEVEL_ELEMENT_INDEX+1));
   __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "App DTT Model handle: %d\n",
         m_dtt_server_1.server.model_handle);


   ERROR_CHECK(app_onoff_scene_context_set(&m_onoff_server_0, &m_scene_server_1));
   __LOG(LOG_SRC_APP, LOG_LEVEL_INFO, "App Scene Model Handle: %d\n", m_scene_server_1.scene_setup_server.model_handle);
```

Figura 7.17: Inicialización modelo servidor

Para demostrar que estos modelos servidores adicionales son necesarios se ha hecho una breve prueba, comentándolos y cargando de nuevo el programa. El resultado ha sido el siguiente:

```
Download successful
Stopped by vector catch
```

```
uint32_t app_level_value_restore(app_level_ 0x0004599E
void mesh_events_handle(const nrf_mesh_e 0x000441DA
void event_handle(const nrf_mesh_evt_t* p_ 0x000325A6
```

Figura 7.18: Error por no incluir servidores adicionales

- **Publicación como modelo cliente**

  La publicación de datos de los modelos clientes (según el modelo usado) se hará por medio de otra función que requiere de parámetros entre los que destaca el valor, y el tid (*Transaction ID*).
  En el caso del modelo cliente onoff es:

```
generic_onoff_set_params_t set_params; //Estrutura del modelo cliente On-OFF
model_transition_t transition_params;
static uint8_t tid = 0; //Nº de transaccion

set_params.on_off = OnOFF; //Valor que se enviará
set_params.tid = tid++;
transition_params.delay_ms = APP_ONOFF_DELAY_MS;
transition_params.transition_time_ms = APP_ONOFF_TRANSITION_TIME_MS;

status = generic_onoff_client_set_unack(&ClienteOnOFF,
                                        &set_params,
                                        &transition_params,
                                        APP_UNACK_MSG_REPEAT_COUNT);
```

Figura 7.19: Publicación *onoff*

Para el modelo cliente *dimming* es:

```
uint32_t status = NRF_SUCCESS;

//Parametros del modelo
static generic_level_set_params_t set_params = {0};
model_transition_t transition_params;

set_params.level = dimmingActualValue;  //Valor publicado PWM y calculado

set_params.tid++;

//Se establecen los parametros de delay y transicion
transition_params.delay_ms = APP_LEVEL_DELAY_MS;
transition_params.transition_time_ms = APP_LEVEL_TRANSITION_TIME_MS;


//Se realiza publicacion
(void)access_model_reliable_cancel(ClientDimming[0].model_handle);
status = generic_level_client_set(&ClientDimming[0], &set_params, &transition_params);
```

Figura 7.20: Publicación *dimming*

Como se puede observar el procedimiento es muy similar, solo varía el valor a enviar.

- **Escritura y lectura en modelos servidores**
  La escritura y lectura en los modelos servidores, solicitada por los clientes, se realiza por medio de funciones *call-back* definidas previamente en las instancias de los modelos servidores tal y como se mostró en las figuras 7.15 y 7.14. Su comprensión es muy simple, cada vez que se escribe o se lee en el modelo se accionan y simplemente se asigan un valor. Se muestra a continuación un ejemplo de código de uno de los nodos para el modelo *dimming*.

```
/*Callback de escritura HW */
static void app_level_server_set_cb(const app_level_server_t * p_server, int16_t present_level)
{
  if(estadoLuz ==1 ){
      m_pwm0_present_level = present_level;
      dimming_value = m_pwm0_present_level;
      pwm_utils_level_set(&m_pwm, m_pwm0_present_level);
    }

}

/*Callback de escritura HW */
static void app_level_server_get_cb(const app_level_server_t * p_server,  int16_t * p_present_level)
{
  if(estadoLuz==1){
    *p_present_level = m_pwm0_present_level;
   }

}
```

Figura 7.21: Escritura y lectura modelo servidor

Como se ha podido ver en la figura 7.21, se asigna únicamente el valor que recibe como parámetro la función *call-back* de escritura en caso de que la luz esté encendida (este valor se corresponde con el enviado por el modelo cliente). Posteriormente este valor se le pasa a la función ***pwm_utils_level_set(valor)*** que actualiza la PWM de la luz con el valor recibido.

NOTA: El valor enviado por el modelo dimming es un INT16 por tanto está comprendido entre -32768 y 32767.

- **Restauración del valor modelo servidor**

  En ocasiones puede darse la situación que cuando se utiliza una red *mesh*, un nodo se quedé sin batería y por tanto deje de funcionar. En estos casos, un handler se encarga de recuperar el último que tuviera antes de que se cortase la alimentación. Además se mantiene también la configuración, guardando la dirección de los modelos de los nodos.

  El valor se restaura gracias al handler añadido en la función ***start()*** con la única condición de que las variables escritas en los modelos deben de ser estáticas. Las partes de código correspondientes a la restauración del valor se han reunido en una única figura para no poner múltiples ilustraciones:

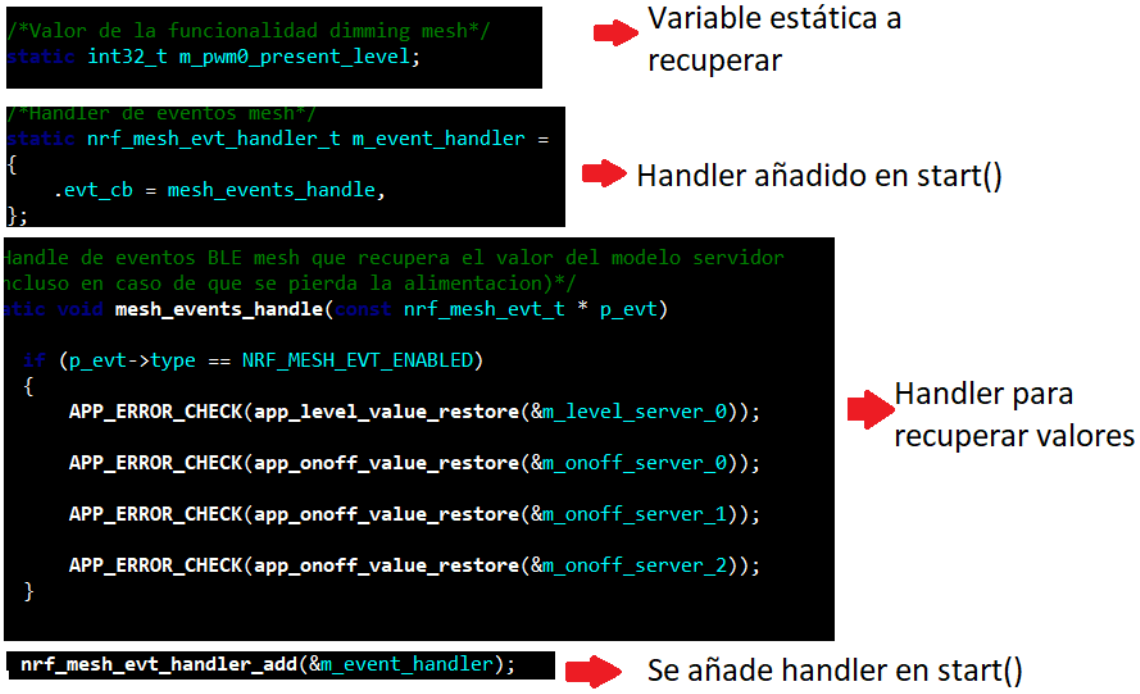

Figura 7.22: Restauración valor modelo

- **Servicio y características BLE**

  Otra parte reseñable del código es la implementación del servicio BLE. En este caso se ha partido de uno realizado inicialmente al que se la han ido haciendo modificaciones para adaptarlo al sistema, añadiendo las características que eran necesarias. Al igual que ocurría con los modelos servidores, los servicios BLE requieren para su inicialización de una instancia. En este caso la instancia es ***BLE_LBS_DEF(TFG_SERVICE)*** que además de definirla hay que habilitar su uso en el *SDK_config.h*. Posteriormente, la inicialización se realiza en la función mencionada inicialmente, ***initialize()***, siendo concretamente ***services_init()***. Con ello arranca el servicio y todas las características que se deseen incluir. Al ser también un código bastante extenso se va a indicar únicamente como crear características, añadirlas y leerlas.

  En ***ble_tfg_service_init(&TFG_SERVICE, &init)*** se añaden las características que se quieran, concretamente gracias a la función ***characteristic_add***, definiendo en ***ble_lbs.h*** el UUID de la característica y una serie de estructuras en donde se guarda el valor. También se indica si la característica es de lectura, notificación...

```c
uint32_t ble_tfg_service_init(ble_lbs_t * p_lbs, const ble_lbs_init_t * p_lbs_init){
    uint32_t              err_code;
    ble_uuid_t            ble_uuid;
    ble_add_char_params_t add_char_params;

    // Se añade el servicio.
    ble_uuid128_t base_uuid = {TFG_UUID_BASE};
    err_code = sd_ble_uuid_vs_add(&base_uuid, &p_lbs->uuid_type);
    VERIFY_SUCCESS(err_code);

    ble_uuid.type = p_lbs->uuid_type;
    ble_uuid.uuid = TFG_UUID_SERVICE;

    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY, &ble_uuid, &p_lbs->service_handle);
    VERIFY_SUCCESS(err_code);

    /**Caracteristicas exclusivas MICROBIT 047**/
    //Caracteristica de  % luz activo (luz3W) BLE (Lectura + notificacion)
    memset(&add_char_params, 0, sizeof(add_char_params));

    add_char_params.uuid             = TFG_INTENSIDAD_LUZ_SERVICE;
    add_char_params.uuid_type        = p_lbs->uuid_type;
    add_char_params.init_len         = sizeof(uint8_t);
    add_char_params.max_len          = sizeof(uint8_t);
    add_char_params.char_props.read  = 1;
    add_char_params.char_props.notify = 1;

    add_char_params.read_access  = SEC_OPEN;
    add_char_params.cccd_write_access = SEC_OPEN;
    err_code = characteristic_add(p_lbs->service_handle, &add_char_params, &p_lbs->tfg_intensidadLuz_handles);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }
```

Figura 7.23: Se agregan las características

Finalmente, para la lectura de los valores BLE es necesario crear otra función, indicando como parámetros el tipo, el dato, la longitud y el manejador de la característica. Además es necesario hacer ***return sd_ble_gatts_hvx(conn_handle, &params);*** para que el valor se actualicé. Esta función será invocada en el main cuando proceda.

```c
/*Funcion encargada de actualizar y transmitir el valor por BLE*/
uint32_t ble_intensidadLuz_service(uint16_t conn_handle, ble_lbs_t *p_lbs, uint8_t cycle){

  ble_gatts_hvx_params_t params;
  uint16_t len = sizeof(cycle);

  memset(&params,0,sizeof(params));
  params.type = BLE_GATT_HVX_NOTIFICATION;
  params.handle = p_lbs->tfg_intensidadLuz_handles.value_handle; //Handle de la caracteristica
  params.p_data = &cycle;
  params.p_len = &len;

  //Transmitimos y actualizamos el dato
  return sd_ble_gatts_hvx(conn_handle,&params);
}
```

Figura 7.24: Lectura y actualización de la característica

- Una última parte reseñable del código es el **cálculo de la distancia** debido a que es necesario seguir el datasheet del sensor GP2Y0A21YK0F[27]. De este modo, se ha seguido el comportamiento por medio del cálculo de rectas sucesivas.

```
/**Funcion encargada de calcular la distancia segun la tension recibida acorde a la grafica del sensor de distancia del datasheet**/
void CalcularDistancia(float value_sensor_distance){
  if(value_sensor_distance>=2.5){
            distancia=(value_sensor_distance -2.2)/0.1 +10;
      }
      else if (value_sensor_distance<2.5 && value_sensor_distance>=2){
            distancia= 20 + (2.5-value_sensor_distance)/0.05;
      }
      else if (value_sensor_distance <2 && value_sensor_distance>=1.5){
            distancia=(30) + (2-value_sensor_distance)/0.05;
      }
      else if (value_sensor_distance<1.5 && value_sensor_distance>=1.25){
            distancia=(40)+(1.5-value_sensor_distance)/0.025;
      }
      else if (value_sensor_distance<1.25 && value_sensor_distance>=1){
            distancia=(50)+(1.25-value_sensor_distance)/0.025;
      }
      else if (value_sensor_distance<1 && value_sensor_distance>=0.9){
            distancia=(60)+(1 - value_sensor_distance)/0.01;
      }
      else if (value_sensor_distance <0.9 && value_sensor_distance >= 0.8){
            distancia = (70) +(0.9-value_sensor_distance)/0.01;
      }
      else if (value_sensor_distance <0.8 && value_sensor_distance >= 0.7){
            distancia = (80) +(0.8-value_sensor_distance)/0.01;
      }
      else if (value_sensor_distance <0.7 && value_sensor_distance >= 0.6){
            distancia = (90) +(0.7-value_sensor_distance)/0.01;
      }
      else if (value_sensor_distance <0.6 && value_sensor_distance >= 0.5){
            distancia = (100) +(0.1-value_sensor_distance)/0.01;
      }
      else if (value_sensor_distance <0.55 && value_sensor_distance >= 0.5){
            distancia = (110) +(0.55-value_sensor_distance)/0.005;
      }
      else if (value_sensor_distance <0.5 && value_sensor_distance >= 0.4){
            distancia = (115) +(0.5-value_sensor_distance)/0.005;
      }
      else if (value_sensor_distance<0.4){
            distancia=150;
      }
}
```

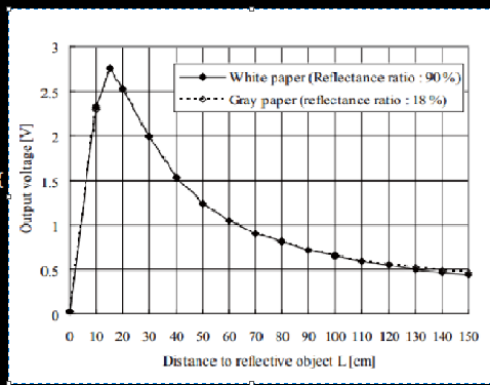

Figura 7.25: Gráfica sensor GP2Y0A21YK0F

****Algunos aspectos importantes que hay que mencionar para que funcione el sistema son****

- 1.- Para que todos los módulos se puedan usar tienen que estar habilitados en el ***SDK_config.h***.

- 2.- Para que BLE funcioné en la micro:bit es necesario cambiar el reloj del SoftDevice a través de la constante ***NRF_SDH_CLOCK_LF_SRC*** 2 ó 1 para la nRF52833DK.

- 3.- Para lograr el funcionamiento de un nodo como LPN además de ***mesh_lpn_init()*** hay que usar la constante ***MESH_FEATURE_LPN_ENABLED 1*** ubicada en el archivo de configuración de la *mesh*, *nrf_mesh_config_app.h*. Lo mismo ocurre para que funcionen como nodos amigo, ***MESH_FEATURE_FRIEND_ENABLED 1***.

- 4.- Para que funcionen los modelos, la constante ***ACCESS_ELEMENT_COUNT*** debe ser igual como mínimo al numero de modelos en *nrf_mesh_config_app.h*.

- 5.- Para que *BLE mesh* y BLE puedan funcionar de manera paralela se debe de incluir en el archivo de configuración de la *mesh*, nrf_mesh_config_app.h las constantes
  MESH_FEATURE_PB_GATT_ENABLED 1
  MESH_FEATURE_GATT_PROXY_ENABLED 1

****

7.2.2 Programación Node-Red

La programación en Node-Red se realizará para transmitir los datos BLE por MQTT a una página web gracias al uso de un broker público. Para ello, aunque su programación no será tan exhaustiva como la de los microcontroladores, se indicarán cuales han sido las tareas realizadas. De igual manera, el código de Node-Red estará accesible en el capítulo 11. *Node-RED* cuenta con una librería (cuya instalación se explicó en el capítulo 6) que permite extraer los datos de los servicios y características BLE de un dispositivo. Inicialmente este nodo, busca los dispositivo BLE a los que se puede conectar. Para ello, basta con escanear, pinchando en la parte remarcada de la figura 7.26. Como resultado, una vez que se seleccione el dispositivo deseado se obtiene su MAC y todas sus características, incluyendo de qué tipo son (lectura, escritura, notificación...).

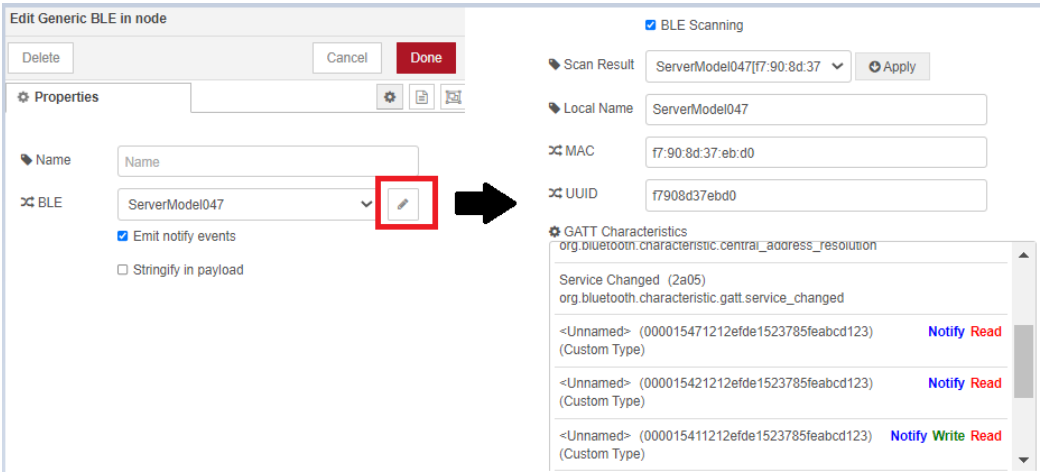

Figura 7.26: Ejemplo conexión BLE

Una vez conectado con el dispositivo BLE en *Node-RED*, es hora de extraer los datos de las características. Para ello, se utiliza un bloque de función. En él, *Node-RED* nos permite declarar variables que se inicializan una sola vez cuando se ejecuta el nodo. Estas variables, que se declaran como globales, se utilizan para tenerlas como *back-up* en caso de que fallé dado que se actualizarán continuamente.

Figura 7.27: Variables globales para back-up

Posteriormente, ya es en el desarrollo de la función en donde extraemos los valores de las características a partir del dato estructurado que nos devuelve el nodo BLE. Tras ello, se actualiza el valor de la variable global y finalmente todos los datos extraídos se guardan en un *JSON* que es el que se envía por MQTT (tal y como se muestra en la parte remarcada). En la figura siguiente se muestra el proceso explicado, correspondiéndose con el nodo **"Datos Servicios Microbit047"** de la figura 7.32:

```
//Guardamos en variables globales para tenerlas como backup
try{
    estadoLuz  = msg.payload.characteristics["000015411212efde1523785feabcd123"][0];
    global.set("estadoLuz047", estadoLuz);
}
catch(error){
    estadoLuz = global.get("estadoLuz047");
}

try{
    estadoNodo = msg.payload.characteristics["000015421212efde1523785feabcd123"][0];
    global.set("estadoNodo047", estadoNodo);
}
catch(error){
    estadoNodo = global.get("estadoNodo047");
}

try{
    cantidadLuz = msg.payload.characteristics["000015471212efde1523785feabcd123"][0];
    global.set("cantidadLuz047",cantidadLuz);
}
catch(error){
   cantidadLuz = global.get("cantidadLuz047");
}

/***********/

estadoNodo_string = estadoNodo == 0 ? 'OFF' : 'ON';
estadoLuz_string = estadoLuz == 0 ? 'OFF' : 'ON';

var datosLectura = {estado_Nodo_047: estadoNodo_string,
                    estado_Luz_047:estadoLuz_string,
                    intensidadLuz_047: cantidadLuz
};

var d = new Date();

datosLectura.now = d.getTime();
msg.payload = JSON.stringify(datosLectura);

return msg;
```

Figura 7.28: Extracción BLE

Los datos extraídos BLE de la figura 7.28 son retornados en forma de *JSON* y también son guardados en otra variable global para tenerla como *back-up* por medio de otro nodo.

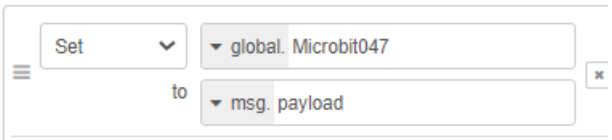

Figura 7.29: V.Global Nodo

Este procedimiento es análogo para todos los nodos que conforman la red en malla. Extraídos todos los datos y almacenados en diferentes variables globales ahora es momento de transmitirlos por MQTT a la página web. Para ello, *Node-RED* cuenta con nodos de publicación MQTT. Sin embargo, antes de transmitirlos, es necesario agruparlos en otra variable y poner datos en un formato correcto para que el broker pueda trabajar correctamente con ellos, agrupándolos en un único *JSON*. Esta parte se corresponde con el nodo **"Agrupación Datos"** de la figura 7.33.

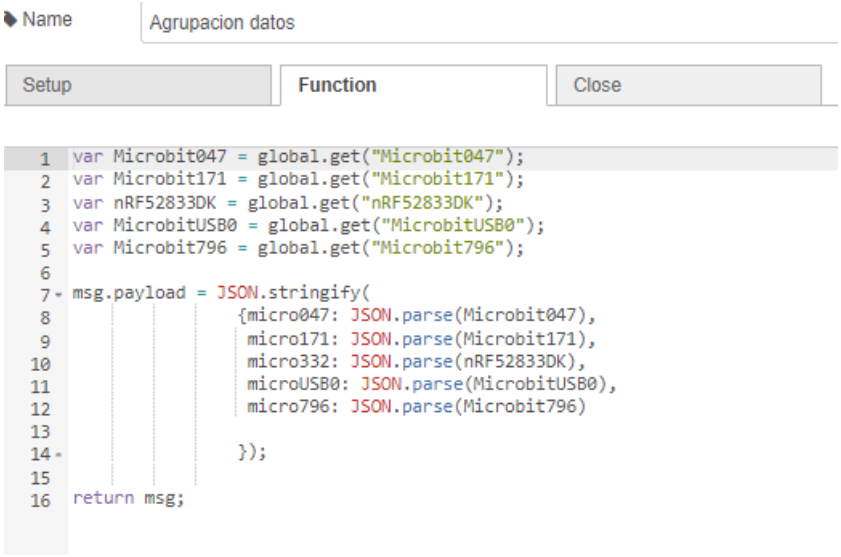

Figura 7.30: Datos agrupados - Nodo función "Agrupación Datos"

Finalmente el mensaje retornado por esta función es enviado a la página web por MQTT. El nodo MQTT también es necesario configurarlo, indicando la dirección del servidor(en este caso público) y el tópico.

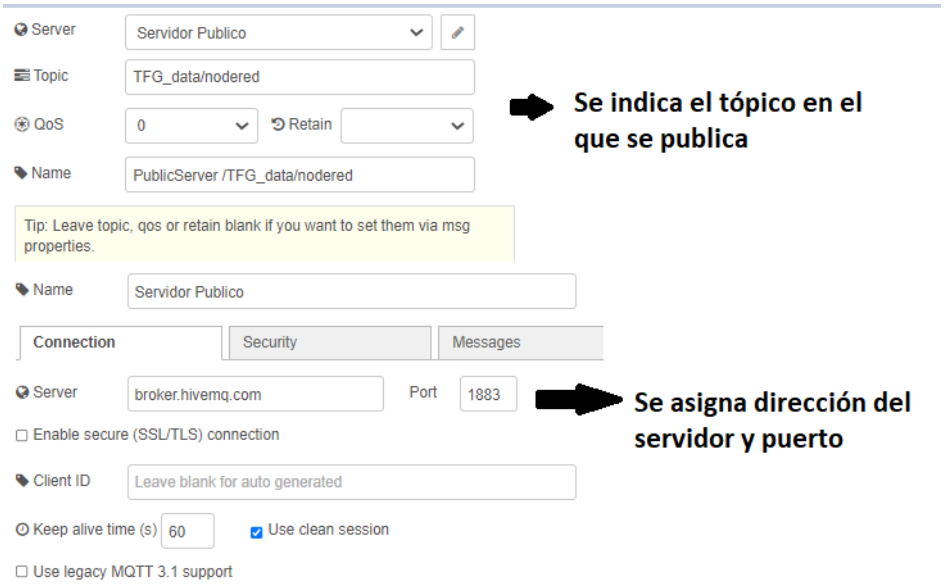

Figura 7.31: Configuración nodo MQTT

La implementación del flujo *Node-RED* en su totalidad se muestra a continuación, distinguiéndose dos partes; la extracción de datos BLE y la publicación a MQTT.

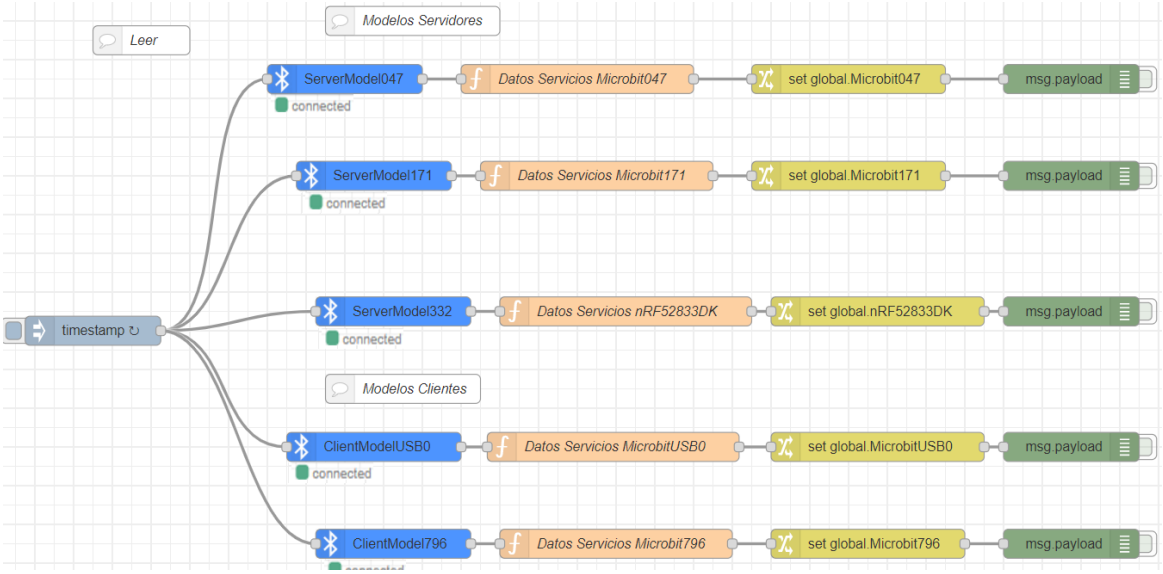

Figura 7.32: Flujo *Node-RED*. Extracción de datos BLE

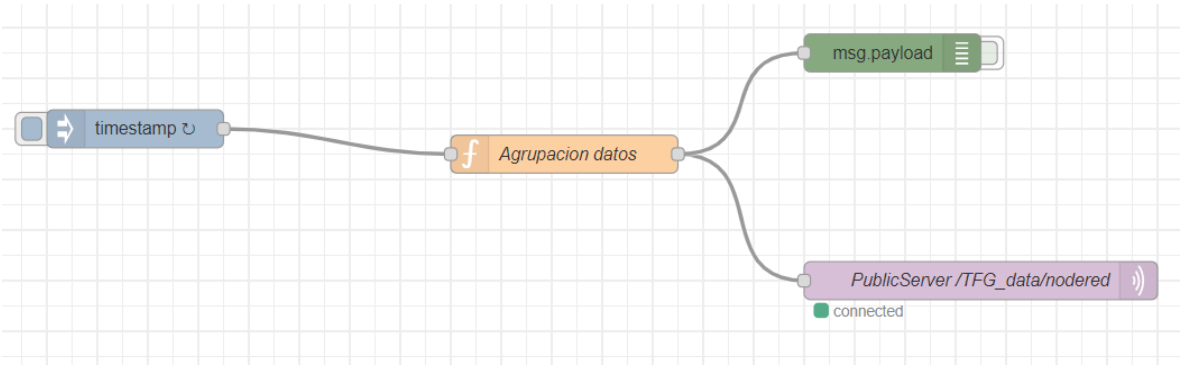

Figura 7.33: Flujo *Node-RED*. Publicación MQTT

Con ello, la parte de *Node-RED* quedaría finalizada y en el momento en que un usuario se suscriba (en este caso uno mismo), tendrá acceso a estos datos, pudiendo darles un formato a través de una página tal y como se explica a continuación.

7.2.3 Programación página web

Por último es necesario mencionar el aspecto de la parte de la página web, desarrollada con *Javascript* y *HTML*. Tiene como objetivo realizar la representación de datos recibidos por MQTT implementada por *Node-RED*. Para ello es necesario que un hipotético cliente se suscriba al tópico. Para poder usar MQTT es necesario importar las librerías:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js" type="text/javascript"></script>
```

Figura 7.34: Librería MQTT

Importada la librería ya se puede crear un supuesto cliente que al suscribirse al cliente reciba los datos de los servicios BLE publicados en *Node-RED*. Los pasos son:

- Declaración de variables de conexión como el puerto, el host y el cliente.

```
window.onload = function () {
    /*Variables de conexion*/
    var reconnectTimeout = 2000; /*Reconexion timeout*/
    var host ="broker.hivemq.com"; /*Broker*/
    var port = 8000;

    /*Topicos a los que suscribirse*/
    var client ={};
    var myTopic="TFG_data/nodered";
    console.log("On load");

    //Se crea una instancia de un cliente MQTT que recibe como parametros el broker, el puerto(TCP) y el cliente
    client = new Paho.MQTT.Client(host, port, "clientId");
```

Figura 7.35: Declaración variables conexión

- El cliente creado en la figura 7.35 dispone de 2 funciones de call-back en caso de que se pierda conexión y lleguen mensajes. Finalmente inicia la conexión, suscribiéndose al tópico.

```
//Funciones callback de conexion perdida y mensaje recibido
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
//El cliente se conecta por MQTT a un topico por medio de una funcion callback
client.connect({onSuccess:onConnect});
```

Figura 7.36: Funciones *call-back* y conexión

- Se desarrollan las funciones *call-back* y la conexión mencionadas anteriormente.

```
/*Funcion encargada de suscribir a los clientes a un topico*/
function onConnect() {
    console.log("En conexion");
    client.subscribe(myTopic);
}

/*Funcion callback de conexion perdida. En caso de que no llegue nada se reconecta llamando a la funcion call-back*/
function onConnectionLost(responseObject) {
    if (responseObject.errorCode !== 0) {
        console.log("En conexion perdida:"+responseObject.errorMessage);
    }

    client.connect({onSuccess:onConnect});
}
```

Figura 7.37: Desarrollo de *call-back* y suscripción

- Finalmente se desarrolla la función *call-back* de recepción de mensajes, separando los datos del *JSON* en campos por medio de la función ***JSON.parse***. En ella se llaman a los diferentes métodos para actualizar las visualizaciones realizadas en la parte *HTML*.

```
/*Funcion call-back de rececpcion de mensajes*/
function onMessageArrived(message) {
   console.log("Mensaje recibido:"+message.payloadString);

    // convierte en objeto el JSON recibido en el mensaje
    var lectura = JSON.parse(message.payloadString);
   //Se actualizan los datos de las visualizaciones.

   /* Se llaman a los servicios de la cocina - MICROBIT 047*/
    //Leds
    updateEstadoLuzCocina(lectura.micro047.estado_Luz_047);
    updateEstadoNodoCocina(lectura.micro047.estado_Nodo_047);

    //Gauges
    updateTempGaugeCocina(lectura.microUSB0.temperatura_USB0);
    updategasGauge(lectura.microUSB0.cantidadGas_USB0);
    updateIntensidadLuzCocinaGauge(lectura.micro047.intensidadLuz_047);

    //Historico
    chartAddDataCocina(lectura.microUSB0.temperatura_USB0,
                       lectura.micro047.intensidadLuz_047,
                       lectura.microUSB0.cantidadGas_USB0,
                       lectura.micro047.now);

    /************************************/
```

Figura 7.38: Recepción de datos y actualización de visualizaciones parte 1

```
/*Se llaman a los servicios de la entrada - nRF52833DK 332*/
//Leds
updateEstadoLuzEntrada(lectura.micro332.estado_Luz_332);
updateEstadoNodoEntrada(lectura.micro332.estado_Nodo_332);
updateEstadoPuertaEntrada(lectura.micro332.estadoPuerta_332);
updateModoPuertaEntrada(lectura.micro332.modoPuerta_332);

//Gauges
updateTempGaugeEntrada(lectura.microUSB0.temperatura_USB0);
updateDistanciGaugeEntrada(lectura.micro332.distanciaMedida_332);

//Historico
chartAddData(lectura.microUSB0.temperatura_USB0,
             lectura.micro332.distanciaMedida_332,
             lectura.microUSB0.now);

/*****************************************/
```

Figura 7.39: Recepción de datos y actualización de visualizaciones parte 2

```
   /* Se llaman a los servicios del salon - MICROBIT 171*/
   //Leds
   updateEstadoLuzSalon(lectura.micro171.estado_Luz_171);
   updateEstadoNodoSalon(lectura.micro171.estado_Nodo_171);
   updateModoLucesSalon(lectura.micro171.modoLuces_171);

   //Gauges
   updateTempGaugeSalon(lectura.microUSB0.temperatura_USB0);
   updateLuminosidadGaugeSalon(lectura.micro796.Luminosidad_796);
   updateEstadoPersianasGaugeSalon(lectura.micro171.estadoPersianas_171);

   //Historico
   chartAddDataSalon(lectura.microUSB0.temperatura_USB0,
                     lectura.micro796.Luminosidad_796,
                     lectura.micro171.estadoPersianas_171,
                     lectura.micro171.now);
}
```

Figura 7.40: Recepción de datos y actualización de visualizaciones parte 3

Se omite la parte del código *HTML* al ser muy extensa. Sin embargo al igual que todo, en el capítulo 11 está accesible.

El resultado de la página web (sin datos) se visualiza a continuación:

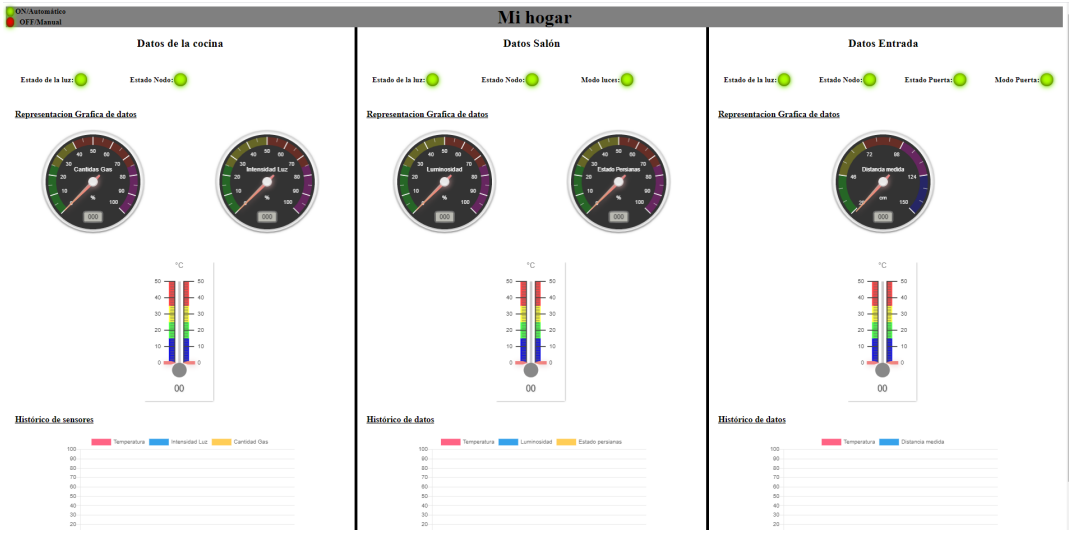

Figura 7.41: Resultado página web

En la figura 7.41 se puede observar tres partes diferenciadas separadas por líneas negras que serían los nodos ubicados en la cocin, el salón y la entrada de una hipotética casa. De esta manera se tiene:

- **Cocina**
  Los 2 leds hacen referencia al estado de la luz y el estado del nodo (en verde ON y en rojo OFF). Por otro lado hay 3 medidores para la cantidad de gas (izquierda), cantidad de luz (derecha) y un termómetro que hace referencia a la temperatura. Finalmente en la parte inferior hay un histórico de valores en una gráfica cuya leyenda es; rosa para la temperatura, azul para la cantidad de luz y amarillo para la cantidad de gas.

- **Salón**
  Los 3 leds hacen referencia al estado de la luz, el estado del nodo (en verde ON y en rojo OFF) y modo luces (en verde es automático y en rojo es manual). Por otro lado hay 3 medidores para la luminosidad (izquierda), ciclo de trabajo de las persianas (derecha) y un termómetro que hace referencia a la temperatura. Finalmente en la parte inferior hay un histórico de valores en una gráfica cuya leyenda es; rosa para la temperatura, azul para la luminosidad y amarillo para el ciclo de trabajo de las persianas.

- **Entrada**
  Los 4 leds hacen referencia al estado de la luz, el estado del nodo, el estado de la puerta (en verde ON y en rojo OFF) y modo puerta (en verde es automático y en rojo es manual). Por otro lado hay 1 medidor para la distancia y un termómetro que hace referencia a la temperatura. Finalmente en la parte inferior hay un histórico de valores en una gráfica cuya leyenda es; rosa para la temperatura, azul para la distancia medida por el sensor con respecto a la puerta.

Capítulo 8

Resultados obtenidos

Dado que se trata de un proyecto que requiere una implementación, su demostración necesita de un vídeo. Debido a esto, se ha tomado la decisión de grabar un vídeo y subirlo a un repositorio que posteriormente se mencionará en el capítulo 11. En total, 2 serán los vídeos que se van a presentar. En el primero se va a mostrar cómo realizar el aprovisionamiento desde cero y el mapeo de direcciones por medio de la app **nRF Mesh**. En el segundo, se muestra el funcionamiento del propio sistema y como los datos se visualizan correctamente en la página web. Sin embargo, lo que si que se puede mostrar es el resultado obtenido de las visualizaciones. Por este motivo, se han realizado diferentes situaciones que demuestran el funcionamiento del sistema.

- **1. Se activan modos automáticos por medio de la micro:bit 761**.
  Debido a esto, los modos cambian (led izquierdo y derecho remarcados pasan a verde), la puerta se abre (led remarcado del medio pasa a verde) si la distancia(dato en azul remarcado en la gráfica derecha) está por debajo de 40cm como es el caso y las persianas se mueven en función de la luminosidad (datos en amarillo y en azul respectivamente remarcados en el gráfico intermedio).

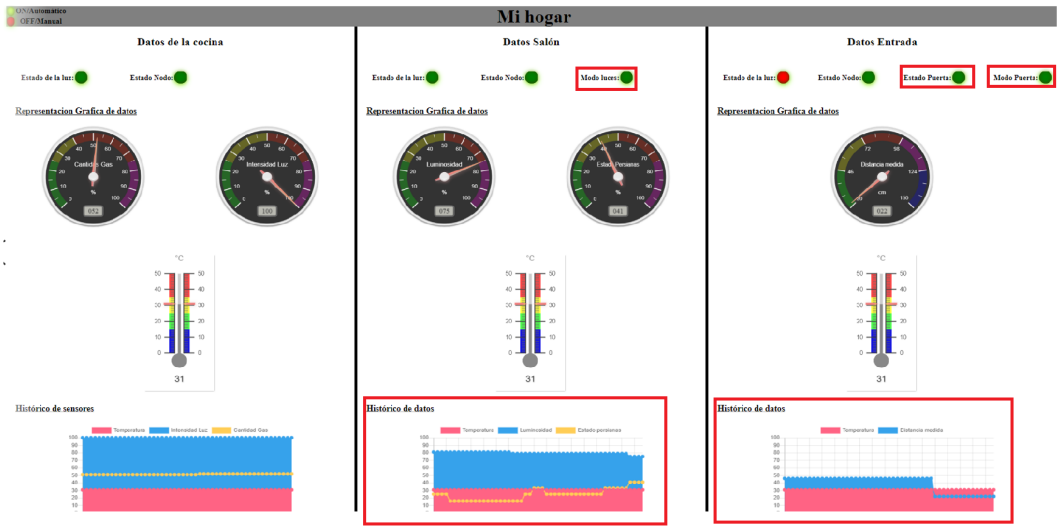

Figura 8.1: Modos automáticos

- **2. Nodos deshabilitados - micro:bit 561 publica a la dirección de grupo**.
  Estados nodos, leds remarcados, pasan a rojo indicando que están deshabilitados.

Figura 8.2: Nodos deshabilitados

- **3. Se activan modos manuales por medio de la micro:bit 761.**
  Los leds remarcados que en la primera imagen, figura 8.1, estaban en verde pasan a rojo.

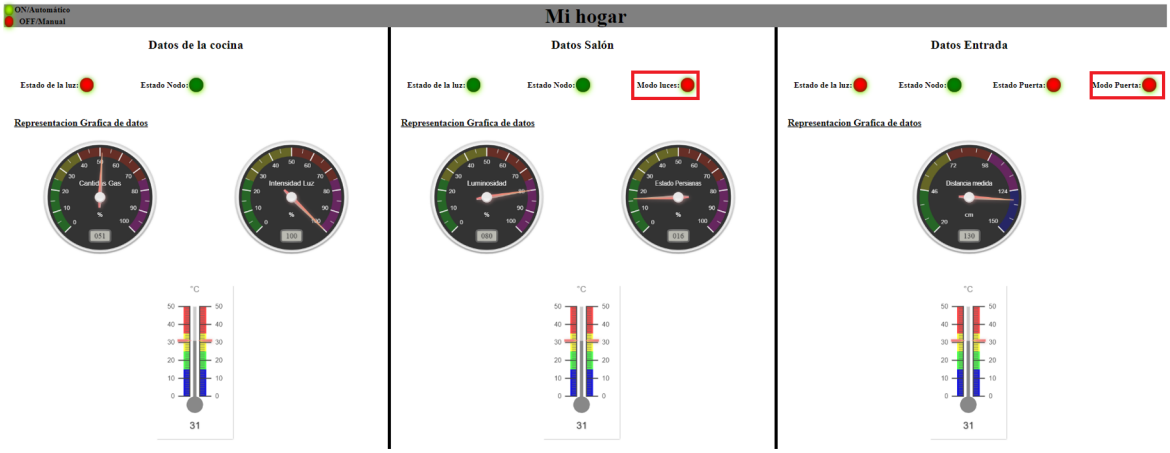

Figura 8.3: Modos manuales

- **4. Varíación de sensores**.
  El sensor de gas(dato en amarillo remarcado en la gráfica izquierda) se ve modificado así como el de distancia y el de luminosidad(dato en azul remarcado en la gráfica intermedia). En este caso la puerta se muestra abierta debido al estado anterior y el ciclo de trabajo de las persianas(dato en amarillo remarcado en la gráfica intermedia) ya no depende de la luminosidad sino del potenciómetro al estar en modo manual.

Figura 8.4: Variación sensores

- **5. Actuación de potenciómetros**.
  En este caso, el funcionamiento de las persianas y luces está en modo manual y el de la puerta están en modo manual (leds rojos remarcados). Por tanto, la luz de la cocina (dato en azul remarcado en la gráfica de la izquierda) y la luz y el ciclo de trabajo de las persianas del salon (dato en amarillo de las gráfica remarcada del medio) varían en función del valor enviado por los potenciómetros.

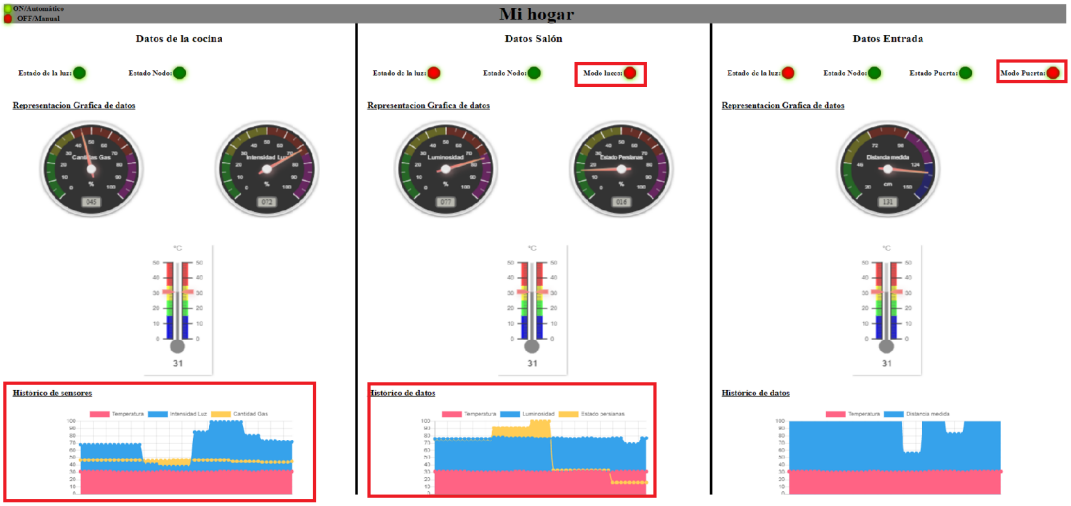

Figura 8.5: Actuación potenciómetros

Con el fin de reforzar la idea de este capítulo, se añaden las trazas Bluetooth y MQTT capturadas con *Wireshark* . De esta manera se puede ver como la comunicación es vía Bluetooth y la visualización de datos se realiza vía protocolo MQTT implementado en *Node-RED*.

Trazas MQTT

Un ejemplo de traza MQTT capturadas por *Wireshark* se puede ver a continuación. En este caso, se corresponde con una traza de publicación hacia la direccion del servidor público MQTT para el tópico o tema "TFG_data/nodered". En la 8.6 se muestra el listado de trazas capturados y en la figura 8.7 se muestra el contenido de la traza remarcada en la figura anterior

No.	Time	Source	Destination	Protocol	Length	Info
107	0.544126	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
292	1.544593	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
640	2.544767	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
770	3.553426	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
1002	4.557408	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
1424	5.570760	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
1708	6.586540	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]
2069	7.598912	192.168.1.148	52.28.71.130	MQTT	607	Publish Message [TFG_data/nodered]

Figura 8.6: Ejemplo traza MQTT

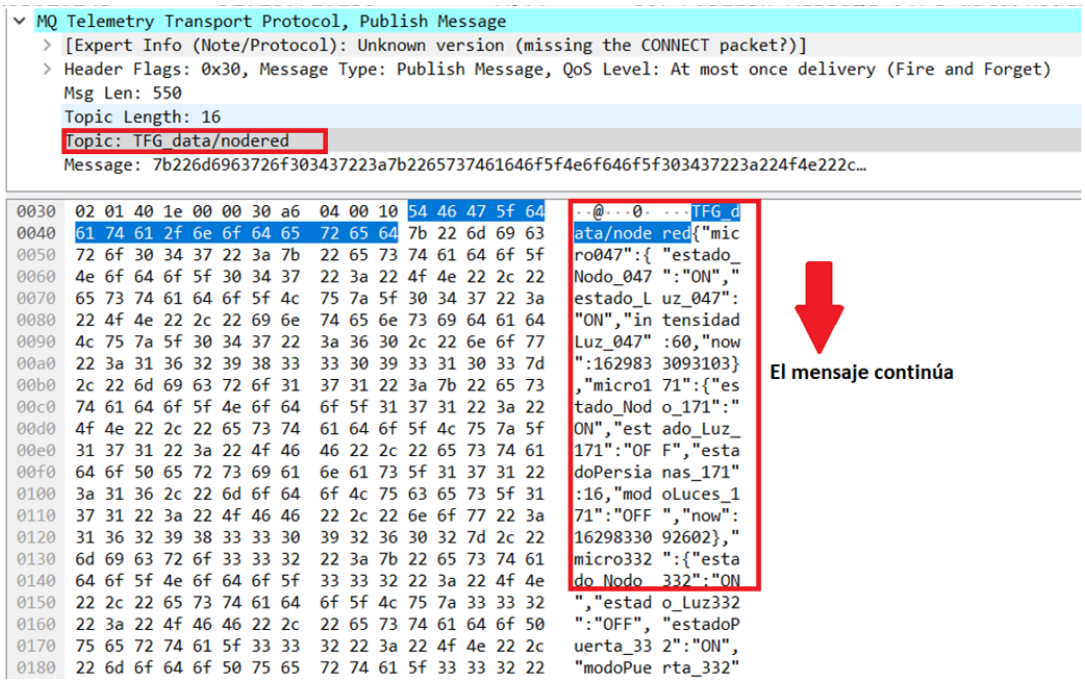

Figura 8.7: Contenido de traza MQTT

Trazas BLE

Para la detección de las trazas Bluetooth ha sido necesario utilizar un *sniffer* que Nordic proporciona al usuario en el siguiente enlace:
`https://www.nordicsemi.com/Products/Development-tools/nrf-sniffer-for-bluetooth-le`.
Este *sniffer* se puede cargar en diferentes microcontroladores como es el *nRF52840-dongle*. Dado que se dispone de él, se ha cargado y se ha instalado el *plugin* en *Wireshark*, siguiendo la documentación proporcionada en[28]. Una vez instalado correctamente todo, en *Wireshark* aparecerá la siguiente interfaz sobre la que capturar paquetes.

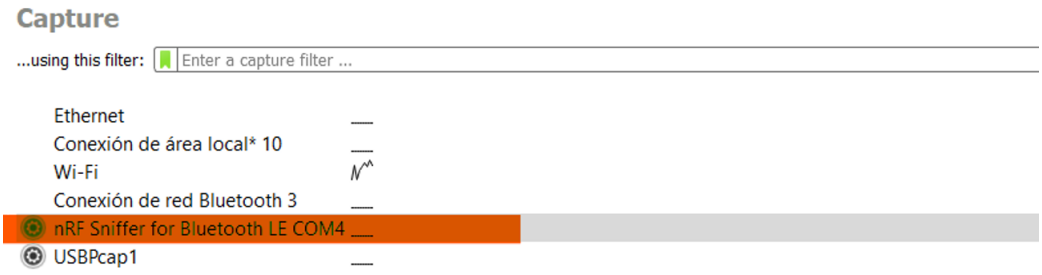

Figura 8.8: Interfaz *Sniffer BLE*

Con el sistema puesto en marcha se analizan los paquetes correspondientes a esa interfaz, encontrando una serie de trazas que se corresponden con los nodos de la red malla BLE diseñada. Se ofrece el listado de direcciones MAC de los nodos así como sus trazas correspondientes:

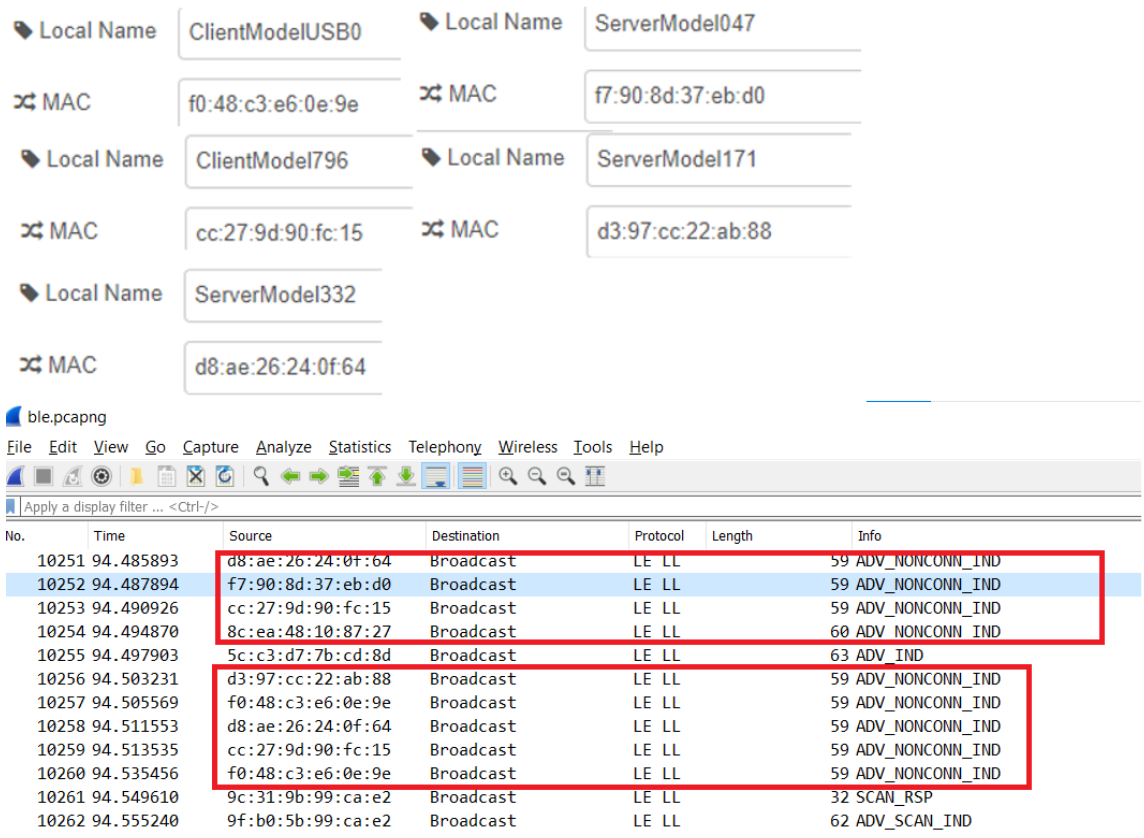

Figura 8.9: Trazas BLE

Analizando uno de los paquetes remarcados de la figura 8.9 se puede observar el contenido del paquete BLE así como parámetros como el canal en el que está transmitiendo el nodo, la dirección del mensaje, la potencia de transmisión (agrupados en flags) entre otros.

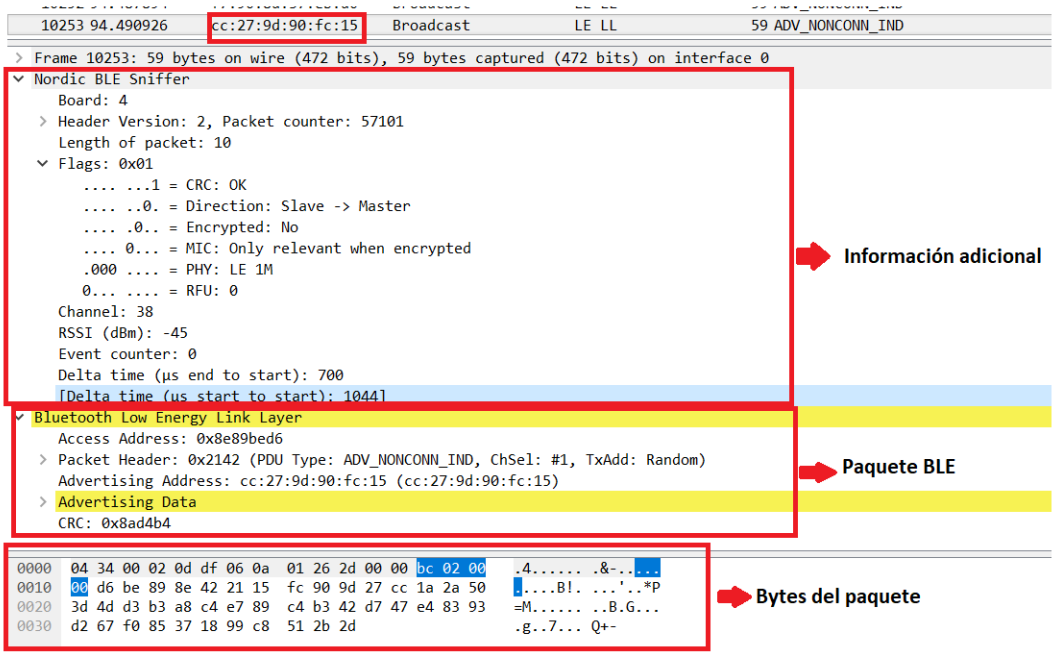

Figura 8.10: Paquete BLE

Capítulo 9

Presupuesto

Para la realización del preupuesto del proyecto se han tenido en cuenta varios aspectos. Por un lado el coste de materiales que componen el trabajo realizado, es decir las placas, elementos HW externos y el USB4.0 mencionado en el capítulo 6. Por otro lado, también se ha tenido en cuenta la mano de obra es decir el coste por el análisis, investigación y estudio del trabajo requerido para su completa realización. Por último se han tenido en cuenta los imprevistos generados durante la ejecución del proyecto.

De este modo el presupuesto se muestra a continuación:

Material	**Nº unidades**	**Coste + IVA/unidad(€)**
micro:bit V2	5 uds.	21,72€
nRF52833DK	1 ud.	49,00€
Servomotor	2 ud.	1,14€
Altavoz + amplificador	1 ud.	2€
Sensor distancia GP2Y0A21YK0F	1 ud.	3€
LDR + PIR + zumbador + Sensor gases	1 ud.	3€
Luces 3 W	3 uds.	0.50€
USB 4.0	1 ud.	11,61€
Mano de obra(h)	**Coste + IVA(€/h)**	
300h totales	15€/h	
Imprevistos	**Coste + IVA (€)**	
Probar con diferentes placas	53,24€	
Total	**4.734,32€**	

Tabla 9.1: Presupuesto

Capítulo 10

Conclusiones y líneas futuras

Conclusiones

Con este proyecto, como se presentó en la introducción, se pretendían abordar una serie de objetivos.

Estos objetivos se basaban principalmente en el aprendizaje de Bluetooth que junto con la utilización del protocolo MQTT y el dominio de los microcontroladores permitían construir un sistema IoT. Como resultado, se ha obtenido una red en malla Bluetooth basada en estos microcontroladores que permite realizar una monitorización de datos gracias el PC que actúa como *gateway*. De este modo se tiene un sistema sólido y de bajo coste que cumple con las bases de todo sistema IoT.

Los conocimientos teóricos adquiridos de Bluetooth, capítulo 2, han servido como base para construir una red en malla Bluetooth en la que además de haber una comunicación entre los microcontroladores también se implementan servicios y características Bluetooth propios de todo sistema IoT. Los valores de estas características y servicios BLE se han plasmado en una página web, permitiendo realizar la monitorización de datos gracias al protocolo MQTT aprendido en el capitulo 3. Todo el trabajo se ha basado en el uso de las tarjetas micro:bit y nRF52833DK con el uso de sus módulos tal y como se presentaron en el capítulo 4. Gracias a ellos y al uso de los entornos de trabajo explicados e instalados en los capítulos 5 y 6 se ha obtenido un sistema robusto y de fácil uso cuya explicación y resultados se muestran en los capítulos 7 y 8. Con ello, y tomando como referencia estos puntos se han conseguido cumplir los objetivos propuestos de este proyecto planteados al inicio, así como la realización de una memoria en donde se han ido enlazado cada una de las partes para lograr una mejor comprensión.

De esta manera, este sistema se podría utilizar tanto en el ámbito personal (en una casa) como en el ámbito de la docencia, demostrando que no es necesario depender de grandes empresas para construir algo de tal calibre, sino que con perseverancia, empeño e interés se puede conseguir.

Líneas futuras

- Uso de sensores de mayor calibre para lograr una mejor precisión así como actuadores adicionales.
- Podría ser interesante el uso de elementos audiovisuales para una monitorización adicional de datos.
- Los datos recibidos en la página web almacenarlos en bases de datos e incluso poder exportar datos en formato CSV o Excel, para un posterior análisis de datos.

Capítulo 11

Manual Usuario

11.1 Descarga del *software* y del proyecto

En este último capítulo del proyecto se pretende guiar al futuro usuario, lector e interesado del proyecto, a como descargar y usar el sistema implementado. Para ello, se ha facilitado tanto el desarrollo del código como los archivos necesarios para personalizar el entorno. En el enlace proporcionado al final del capítulo vamos a encontrarnos los siguientes archivos:

- **Carpeta Anotaciones**
  En ella se ofrecen diferentes notas que se han tenido en cuenta para la correcta implementación del sistema como es por ejemplo como instalar la librería BLE en *Node-RED*. Todo esto se mencionó en el capítulo 6

- **Carpeta Código microControladores**
  En esta carpeta residen los códigos desarrollados para el proyecto. Concretamente están en *nrf_sdk/examples/ble_peripheral*. Ambos SDKs (*nrf_sdk y nrf for mesh*) deben de ser descargados y estar ubicados en el disco local C: tal y como se indicó en el capítulo 6.

- **Carpeta Código *Node-RED***
  En esta carpeta reside todos los flujos llevados a cabo en *Node-RED* para mostrar los datos así como las funciones y personalizaciones realizadas.

- **Carpeta Código Página web**
  En ella se ofrece el código *HTML* y JS de la página web para realizar la monitorización de datos.

- **Vídeo demostración de configuración**
  Con el fin de ayudar al futuro lector se ofrece un vídeo en el que se muestra como configurar la red por medio de la app ***nRF Mesh*** disponible en *iOs* y *Android*.

Una vez descargado todo el material es necesario descargarlo en nuestras tarjetas. Para ello primero hay que compilar como se ve en la figura 11.1 y posteriormente hacer la descarga como se muestra en la figura 11.2.

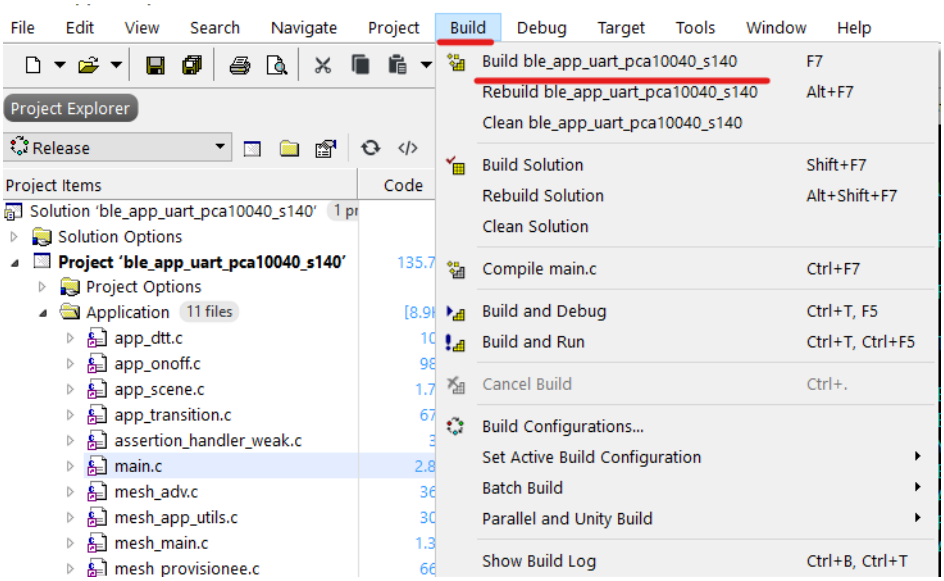

Figura 11.1: Paso 1. Compilación

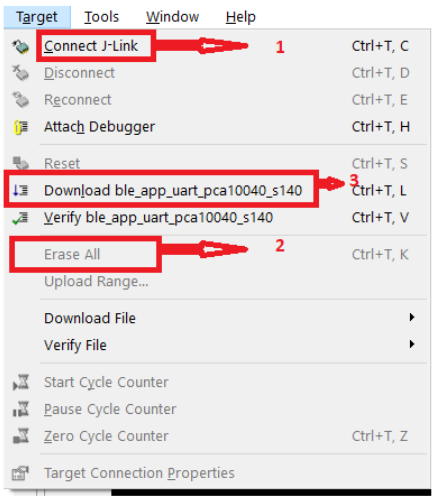

Figura 11.2: Paso 2. Descarga en placa

Tras haber realizado la descarga en placa, una manera de comprobar la correcta implementación es:

- La micro:bit inicialmente tendrá un simbolo "-" en leds apagados.
- La placa nordic tendrá los dos leds inferiores apagados.

El siguiente paso es realizar el proceso de aprovisionamiento y configurar las direcciones de la red. Para ello es necesario descargarse en el móvil la aplicación *nRF Mesh*. Cuando se haya descargado habrá que seguir los siguientes pasos:

- **1. Inicialmente al abrir la aplicación se verá la siguiente figura. Pulsar en el +.**

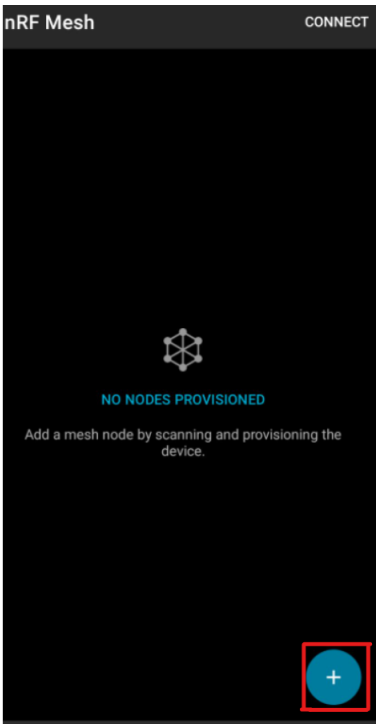

Figura 11.3: Pantalla inicio nRF Mesh App

- **2. Cuando se encuentren los nodos darle a la opción de *Identify* y seleccionar *Provision*. Como puede verse en la figura 11.4, se genera la *App Key* y la dirección *unicast*.**

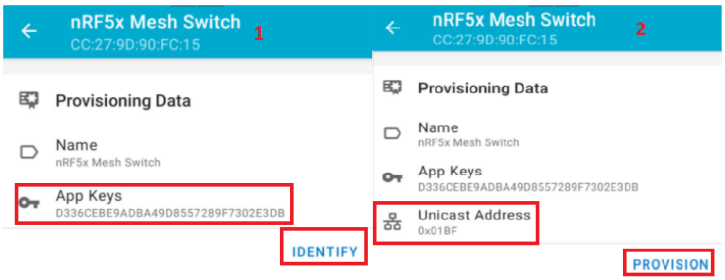

Figura 11.4: Inicio aprovisionamiento

- **3. Seguidamente hay que esperar a que se termine el proceso de aprovisionamiento. En este punto, todas las micro:bit pasarán a tener un símbolo "+" mostrado con leds apagados, indicando que están aprovisionados**

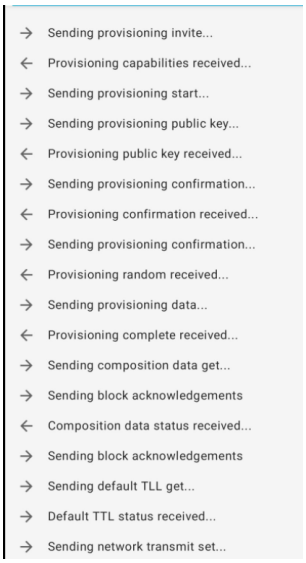

Figura 11.5: Proceso aprovisionamiento

- **4. En el lado del cliente aparecen todos sus elementos y modelos. Por cada elemento hay uno o varios modelos que dan la funcionalidad al sistema.**

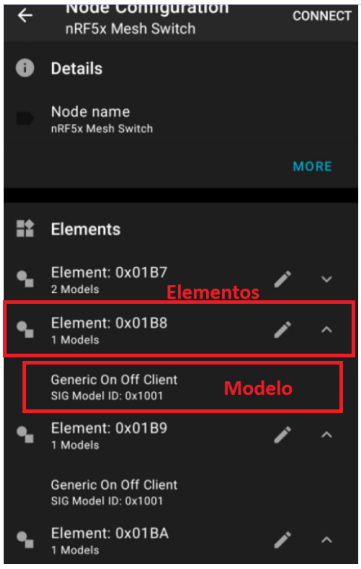

Figura 11.6: Elementos y modelos cliente

- **5. Hay que asignarles las direcciones de publicación unicast de los modelos del servidor (que se pueden observar al terminar el aprovisionamiento). Finalmente darle a *apply*. Es necesario mencionar que para poder hacer esto hay que seleccionar previamente *bind key*, común en todos los modelos.**

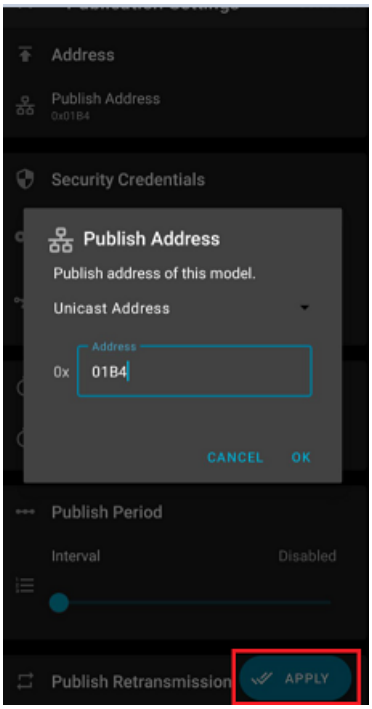

Figura 11.7: Publicación en dirección unicast

- **6. Por otro lado, en los modelos servidor aparecerá algo similar a lo mostrado en la figura 11.6. En ellos, hay que asignar la clave *bind key* igual que en el cliente y posteriormente en caso de que se usen direcciones de grupo, suscribirnos a ellas. En caso contrario no hará falta suscribirse a nada.**

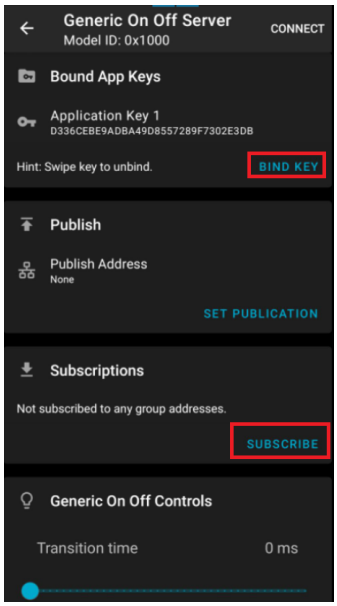

Figura 11.8: Binding y suscripción servidor.

- **7. Finalmente en cada uno de los modelos del servidor hay que darle a *read state* para que la configuración del servidor surja efecto. De este modo el valor *unknown* pasará a tener un valor. En este punto también se pueden configurar parámetros como el delay y los tiempos de transición entre estados entre otros parámetros.**

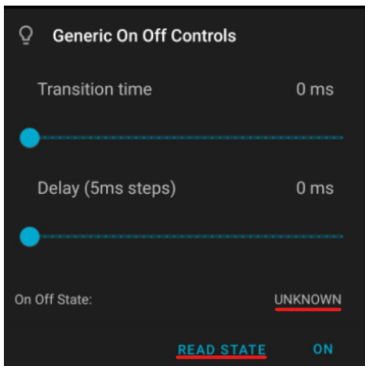

Figura 11.9: Lectura servidor.

NOTA: En caso de que los servidores no se aprovisionen bien a la primera es necesario reiniciar este proceso. Para ello en las micro:bit basta con pulsar el botón B y en la nRF52833DK pulsar el botón 4. Tras esto, volver a los pasos anteriores y repetir de nuevo.

Finalizada la configuración cuyo mapeo de direcciones se muestra en el vídeo, se debe de dar a *disconnect* en la aplicación. Como ya ha sido completada la configuración no será necesario tenerlo abierto. Hecho esto, ya solo queda ver los datos mostrados que se estan transmitiendo por BLE. Para ello, desde cmd abriremos *Node-RED* ejecutando el comando *node-red*. Abierto, ahora se debe de importar los elementos descargados en la carpeta *Carpeta Flujos Node-Red*. Por último darle a *deploy* y abrir el archivo *HTML* para observar los datos transmitidos publicados vía MQTT.

11.2 Modo de uso

Realizado el aprovisionamiento y el mapeo de red de cada uno de los nodos es hora de su utilización. Su funcionamiento es el siguiente:

- **TODOS** los nodos con modelos clientes (561, 796 y USB0) disponen de dos modos de funcionamiento, modo funcionalidad y modo configuración. Se cambian entre sí pulsando el logo (si el led está ON entonces estamos ante el modo configuración). Dentro del modo configuración si se pulsa el botón A se inicia/finaliza amistad con un nodo amigo. Por otro lado si se pulsa B se reinicia el aprovisionamiento, teniendo que realizar los pasos del anterior punto.
- **micro:bit ClientModel 561 - Modo funcionalidad *mesh***
  - Con el botón A envía ON, habilitando el funcionamiento de los nodos micro:bit ServerModel 171, 047 y nRF52833DK ServerModel 332 al publicar en una dirección de grupo.
  - Con el botón B envia OFF, deshabilitando el funcionamiento de los nodos mencionados.
- **micro:bit ClientModel USB0 - Modo funcionalidad *mesh***
  - Con el potenciómetro se regula la luz del led conectado a la micro:bit 047.
  - Con el botón A se enciende o se apaga la luz mencionada, enviando On/OFF.
  - Con el botón B se mueve el servomotor en un sentido u otro, enviando On/OFF a la nRF52833DK ServerModel 332.
  - Un sensor calcula continuamente el porcentaje de gas presente. Si supera un determinado valor envía un ON a un zumbador de la micro:bit 047. Si está por debajo del umbral envía un OFF dejando de sonar.
- **micro:bit ClientModel 796 - Modo funcionalidad *mesh***
  - Con el potenciómetro y siempre que el funcionamiento sea manual se regula una luz y un servomotor conectados a la micro:bit 171.
  - Con el botón A se cambía el modo de luces a automático, enviando ON. De este modo, los elementos no cambian en función del potenciométro sino de la luminosidad calculada por un LDR. Si se vuelve a pulsar se envía OFF, cambiando de nuevo a modo manual.
  - Con el botón B se cambia el modo puerta a automático, enviado un ON. De este modo, el movimiento del servomotor conectado a la nRF52833DK ya no se mueve a partir del botón B de la micro:bit clientModel USB0, sino a partir de un sensor de distancia. Si se vuelve a pulsar se envía OFF, cambiando de nuevo a modo manual y teniendo de nuevo efecto el botón del otro modelo cliente implementado en la micro:bit USB0.-

Todo el código, las diferentes anotaciones y los vídeos de demostración pueden descargarse en el siguiente enlace:
`https://drive.google.com/drive/folders/1z23RA-pVdpSYho4T0MRtvpK5lJeCgoGU?usp=sharing`

Bibliografía

[1] Knud Lasse Lueth, "State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating", 2018. [En línea]. Disponible en: [Accedido: Abril 2021]

[2] Bluetooth SIG, "Bluetooh specifications List". [En línea]. Disponible en: https://www.bluetooth.com/specifications/specs/ [Accedido: Abril 2021]

[3] Fundación Wikipedia INC, "Bluetooth", 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Bluetooth [Accedido: Mayo 2021]

[4] IEEE.org, "Bluetooth Network Encapsulation Protocol (BNEP) Specification", 2001. [En línea]. Disponible en: https://grouper.ieee.org/groups/802/15/Bluetooth/BNEP.pdf
[Accedido: Abril 2021]

[5] Bluetooth SIG, "Learn about Bluetooth. Bluetooth Technology Overview". [En línea]. Disponible en: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ [Accedido: Abril 2021]

[6] Nordic Semiconductor, "Bluetooth Low Energy protocol stack", 2019 [En línea]. Disponible en: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsds_s140%2FSDS%2Fs1xx%2Fble_protocol_stack%2Fble_protocol_stack.html [Accedido: Abril 2021]

[7] Bluetooth SIG, "Core Specification 5.3", 2021. [En línea]. Disponible en: https://www.bluetooth.com/specifications/specs/core-specification/ [Accedido: Mayo 2021]

[8] Bluetooth SIG, "Bluetooth Mesh Networking.An introduction for developers", 2020.
[En línea]. Disponible en: https://www.bluetooth.com/wp-content/uploads/2019/03/Mesh-Technology-Overview.pdf [Accedido: Junio 2021]

[9] S. Safaric and K. Malaric, "ZigBee wireless standard", 2007 [En línea]. Disponible en: https://ieeexplore.ieee.org/document/4127535 [Accedido: Mayo 2021]

[10] V. A. Zyulin, A. N. Semenova, A. K. Brazhnikova and D. A. Burilov, "Features of Building MESH Networks Based on Bluetooth Low Energy 5.1 Technology", 2021 [En línea]. Disponible en: https://ieeexplore.ieee.org/document/9396530 [Accedido: Septiembre 2021]

[11] OASIS, "MQTT Version 5.0", 2019. [En línea]. Disponible en: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.pdf[Accedido: Junio 2021]

[12] Luis Llamas,"¿Qué es MQTT? Su importancia como protocolo IoT", 2019. [En línea]. Disponible en: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/ [Accedido: Junio 2021]

[13] robotsparaniños.com, "micro:bit,la placa reina de las escuelas británicas", 2020. [En línea]. Disponible en: https://www.robotsparaninos.com/microbit-la-placa-reina-de-las-escuelas-britanicas/ [Accedido: Abril 2021]

[14] micro:bit, "Hardware micro:bit V2", 2020 [En línea]. Disponible en: https://tech.microbit.org/hardware/ [Accedido: Abril 2021]

[15] Nordic Semiconductor, "S113 SoftDevice Specification Overview", 2019 [En línea]. Disponible en: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsds_s113%2FSDS%2Fs1xx%2Foverview%2Fproduct_overview.html [Accedido: Mayo 2021]

[16] Nordic Semiconductor, "nRF52833" [En línea]. Disponible en: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52833 [Accedido: Mayo 2021]

[17] Segger, 2021 [En línea]. Disponible en: https://www.segger.com/ [Accedido: Abril 2021]

[18] Node-RED,2021. [En línea]. Disponible en: https://nodered.org/ [Accedido: Abril 2021]

[19] Segger, "BBC micro:bit V2 J-Link OB Firmware", 2021. [En línea]. Disponible en: https://www.segger.com//downloads/jlink#BBC_microbit [Accedido: Mayo 2021]

[20] Node-RED,"Running Node-RED locally", 2020. [En línea]. Disponible en: https://nodered.org/docs/getting-started/local [Accedido: Abril 2021]

[21] NodeJS, 2021. [En línea]. Disponible en: https://nodejs.org/en/ [Accedido: Abril 2021]

[22] Node.js - GitHub, 2021. [En línea]. Disponible en: https://github.com/nodejs/node-gyp [Accedido: Abril 2021]

[23] npmjs.com, 2021. [En línea]. Disponible en: https://www.npmjs.com/package/@abandonware/noble [Accedido: Abril 2021]

[24] Zadig, 2020. [En línea]. Disponible en: https://zadig.akeo.ie/ [Accedido: Abril 2021]

[25] Nordic Semiconductor, "nRF5 SDK v17.0.2", 2021. [En línea]. Disponible en:https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_sdk%2Fstruct%2Fsdk_nrf5_latest.html [Accedido: Mayo 2021]

[26] Nordic Semiconductor, "nRF5 SDK for Mesh v5.0.0", 2021. [En línea]. Disponible en: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_sdk%2Fstruct%2Fsdk_mesh_latest.html [Accedido: Junio 2021]

[27] smart-prototyping.com, "GP2Y0A02YK0F datasheet", 2006. [En línea]. Disponible en: https://www.smart-prototyping.com/image/data/2_components/sensors/101811%20GP2Y0A02YK0F/gp2y0a02yk_e.pdf [Accedido: Mayo 2021]

[28] Nordic Semiconductor, "nRF Sniffer for Bluetooth LE", 2021. [En línea]. Disponible en: https://infocenter.nordicsemi.com/pdf/nRF_Sniffer_BLE_UG_v4.0.0.pdf [Accedido: Julio 2021]

Anexos

.1 Esquemático micro:bit V2

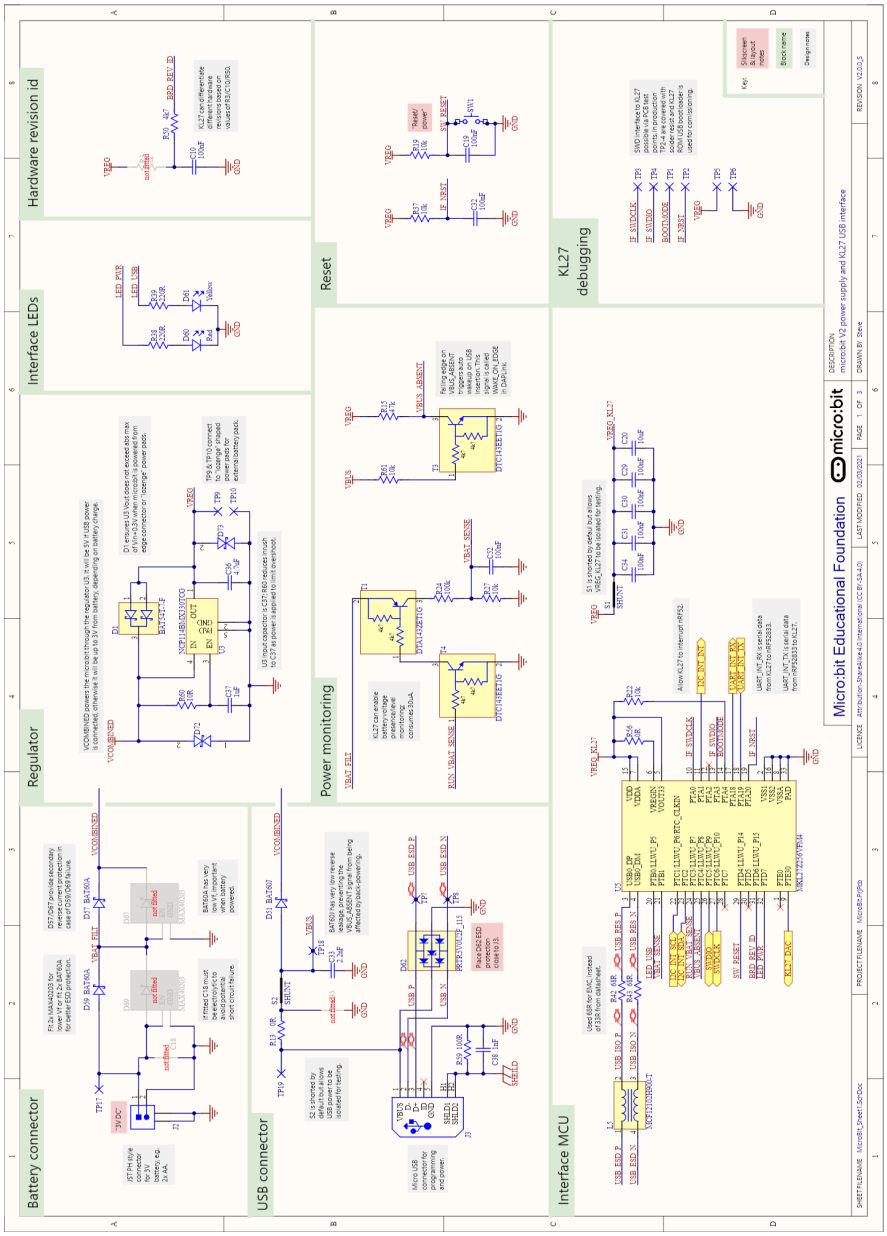

Figura 10: Esquemático micro:bit V2

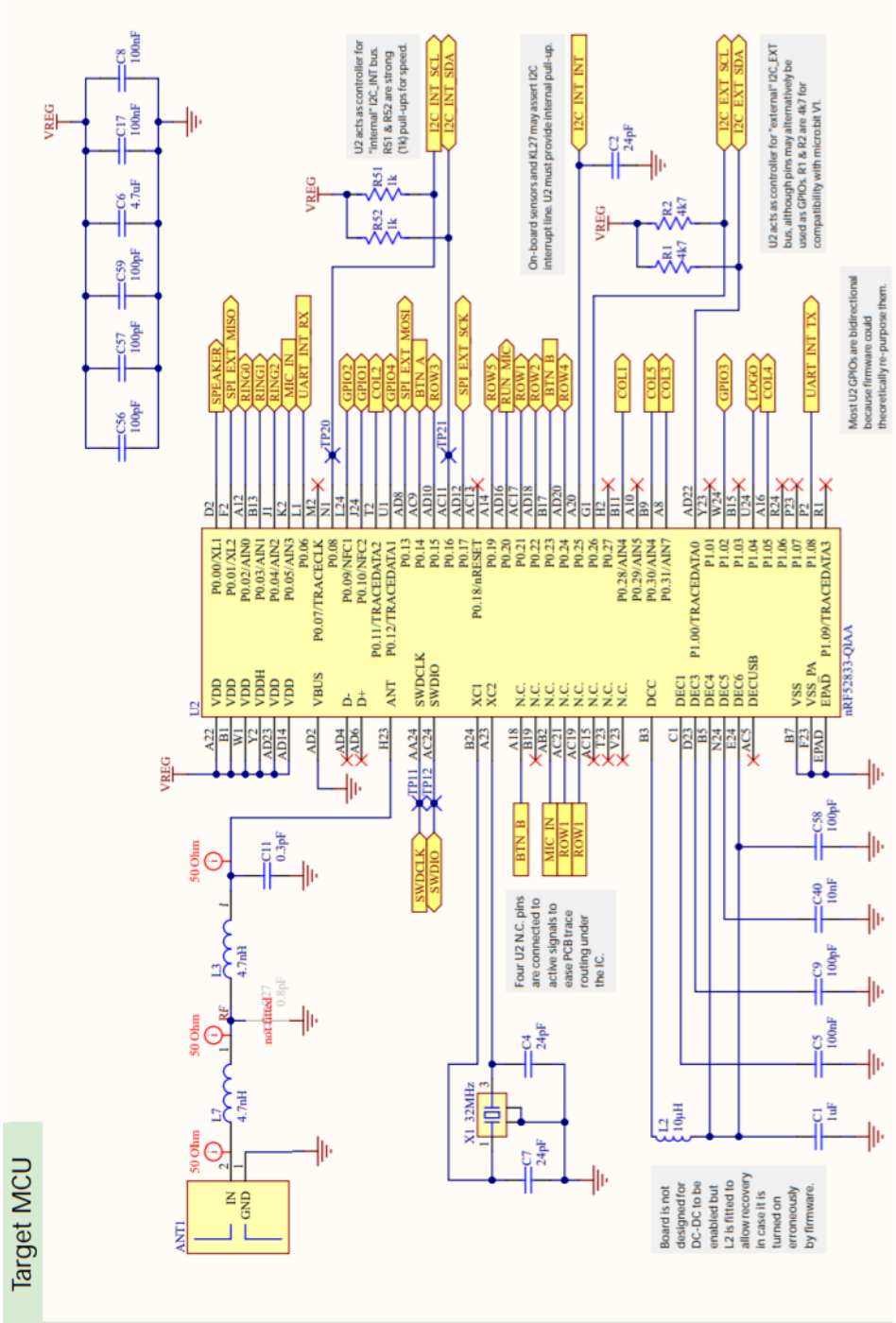

Figura 11: Pinmap micro:bit V2

.2 Esquemático nRF52833DK

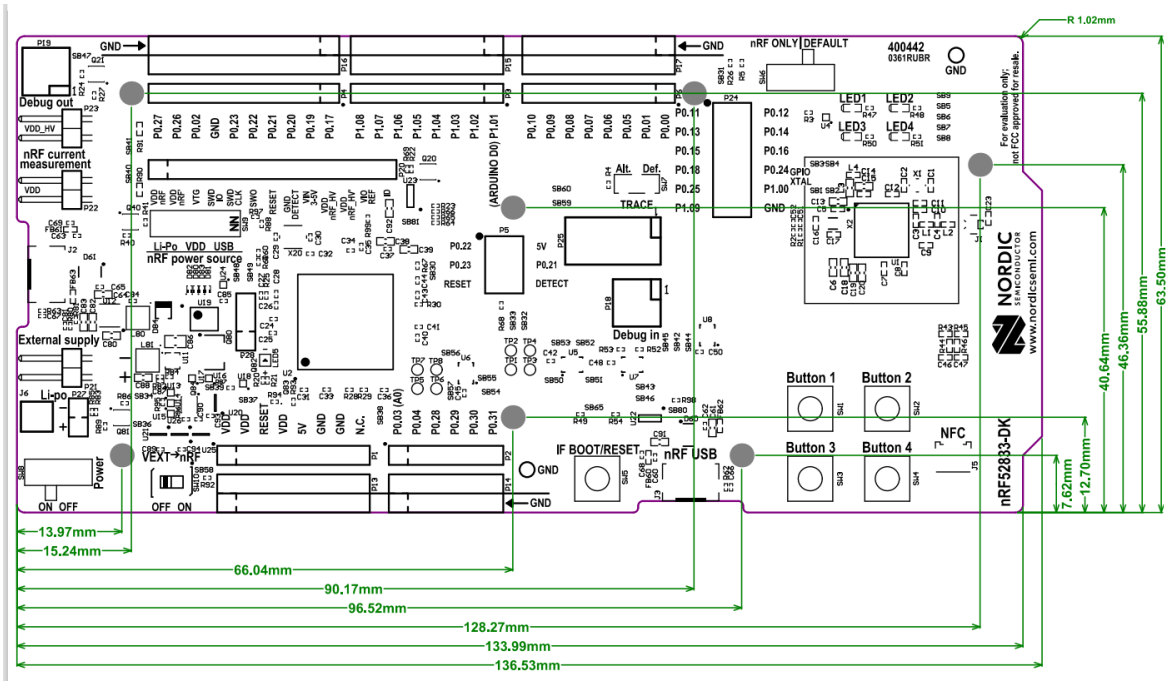

Figura 12: Vista de los pines nRF52833DK

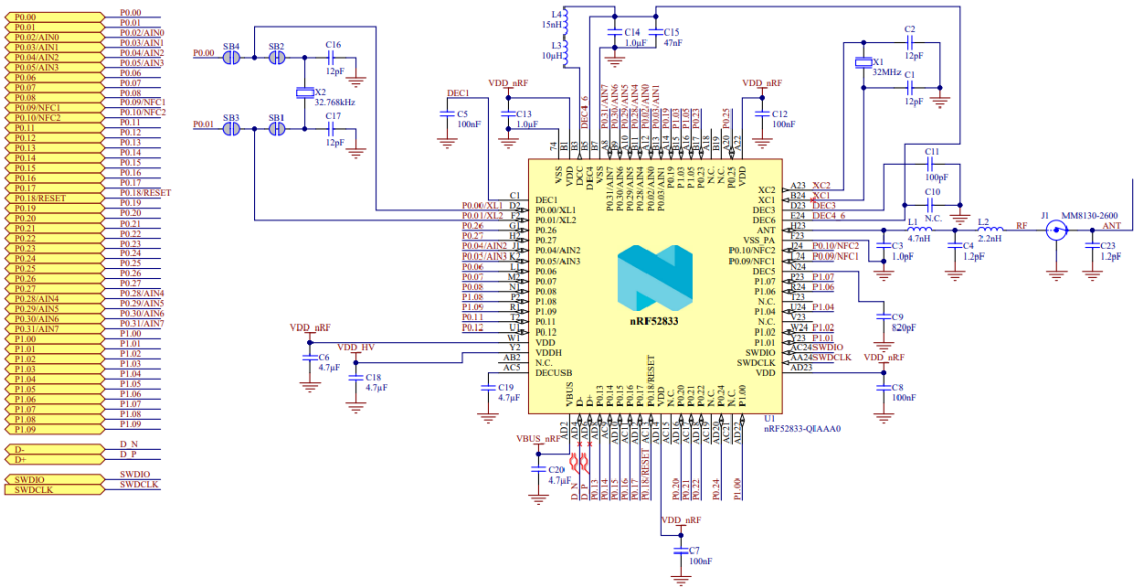

Figura 13: Pinmap nRF52833DK

Universidad de Alcalá
Escuela Politécnica Superior